AFRL-IF-RS-TR-1998-117
Final Technical Report
June 1998

DISTRIBUTED AIR OPERATIONS CENTER

Logicon, Inc.

Pete Gasteiger, Tamara Petroff, Francis A. DiLego

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

APPROVED: [illegible]
FRANCIS A. DILEGO, JR.
Project Engineer

FOR THE DIRECTOR: [illegible]
JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1998	Final Jan 95 - Oct 95

4. TITLE AND SUBTITLE

DISTRIBUTED AIR OPERATIONS CENTER

5. FUNDING NUMBERS

C -	F30602-95-C-0203
PE -	62702F
PR -	5581
TA -	28
WU -	24

6. AUTHOR(S)

Logicon, Inc:	Air Force Research Laboratory:
Pete Gasteiger	Francis A. DiLego, Jr.
Tamara Petroff	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Logicon, Inc.
222 West Sixth Street
San Pedro CA 90731

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/IFSA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1998-117

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Francis A. DiLego, Jr./IFSA/(315) 330-3681

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This final report describes the evolution of the Distributed Air Operations Center (DAOC) architecture over the course of several exercises. Background information on distributed planning in general is included in the appendices. DAOC integrates legacy air campaign planning applications with a distributed infrastructure, providing geographically transparent shared operation. The final prototype architecture uses a server-server paradigm to maintain operability at all DAOC sites while exhibiting robust behaviors through network communications slowdowns and outages. The infrastructure uses a suite of CORBA services to accomplish database replication. The CORBA services are integrated with the applications by adding a CORBA layer between the application and the application database. The CORBA services distribute database transactions to all registered sites. The resulting system provides distributed operation which is robust through low bandwidth and communications interruptions.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Distributed Computing, Air Operations Center, CORBA	88
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Identification

This final report documents all technical work accomplished and information gained during the evolving technology development of the Distributed Air Operations Center (DAOC) system.

1.2 DAOC Overview

The purpose of the DAOC is to demonstrate an operational system that will allow planners at geographically separated sites to work collaboratively to create an Air Tasking Order (ATO) and monitor its execution. In the completed system, an operator at any DAOC site will be able to use legacy Air Operations Center (AOC) applications normally, while having immediate access to planning work done by operators at other DAOC facilities.

DAOC has architecturally evolved in several phases:

- Phase 1 (implemented by DAOC EP1 and demonstrated at JWID 95) consisted of adding a Distributed Computing Environment (DCE) layer to the existing AOC applications to enable distributed operation. Some data transfer capability was included between applications, and the Local Attack Controller (LAC) system was integrated into the AOC suite.

- Phase 2 (implemented by DAOC EP2 and demonstrated at JWID 96) consisted of beginning the migration from the DCE layer of Phase 1 to a Common Object Request Broker Architecture (CORBA) layer, improving the flow of data from one application to another, and improving system fault tolerance to communications interruptions on the Wide Area Network (WAN).

- Phase 3 (implemented by DAOC EP3 and demonstrated at JFACC Jumpstart) consisted of introducing selected Joint Task Force/Advanced Technology Demonstration (JTF/ATD) services into the DAOC suite, and completing the migration from DCE to CORBA.

- Phase 4 (implemented by DAOC EP4 and demonstrated at the DAOC ATD, JWID 97, and Blue Flag 98-1) consisted of upgrading the integrated applications to current versions, and adding an additional CTAPS application.

1.3 DAOC Objectives

The goals of the DAOC system were to investigate distributed collaborative planning techniques for air operations; to integrate Contingency Theater Air Planning System (CTAPS)/Theater Battle Management Core Systems (TBMCS) modules into a distributed planning center; to use CORBA-based technology for distributed processing; and to investigate distributed processing issues for coalition forces.

1.4 Demonstrations

This report spans the completed software tasks for the four-phase development defined for the DAOC system, starting with the technical work demonstrated at JWID 95, and concluding with the demonstration of the Blue Flag 97 capabilities as described above in Section 1.2. The following sections highlight the accomplishments of each phase.

1.4.1 JWID 95

Phase 1 of the DAOC development, demonstrated at JWID 95 and implemented by Evolutionary Prototype 1 (EP1), achieved the following:

- Provided a single database at a central site, Marine Corps Tactical System Support Activity (MCTSSA)
- Provided remote processing at Ft. Franklin and USS Kittyhawk
- CORBA-based Common Database Application Programmer Interface (CDB API) allowed database access over SIPRNet
- CORBA Alert Manager and Corbus-ized Interprocess Manager Host (CRIPMH) service provided distributed application communication
- Provided distributed planning and execution functionality for JWID 95.

1.4.2 JWID 96

Phase 2 of the DAOC development, demonstrated at JWID 96 and implemented by EP2, accomplished the following:

- Provided joint coalition planning and execution for JWID 96
- Provided application databases at every site
- Partitioned data into unique AORs at each planning center
- Synchronized shared data through the CORBA Distributed Update Server (DUS)
- Used NightHawk Guard for data transfer between U.S. and allied planning cells
- DAOC CDB API provided reachback to Rome Laboratory for TNL transfer from RAAP database

1.4.3 JFACC Jumpstart

Phase 3 of the DAOC development, demonstrated at JFACC Jumpstart and implemented by EP3, accomplished the following:

- Provided mission planning and execution for Air Operations
- Provided a Web-based Decision Support System (DSS)
- Integrated JTF ATD services into the DAOC planning environment

1.4.4 JWID 97, DAOC ATD, and Blue Flag 98-1

Phase 4 of the DAOC development, demonstrated at JWID 97 and again at the DAOC ATD and Blue Flag 98-1, accomplished the following:

- Provided forward, rear, and support ("Reachback") capabilities, plus (at JWID 97) airborne and at-sea JFACC positions
- Integrated latest CTAPS air battle planning applications (APS, FLEX, ADS)
- Provided collaboration with coalition partner in Canada (JWID 97)

1.5 Report Organization

This report documents the details of all technical work completed to attain the DAOC objectives described in Section 1.3. This report includes pertinent observations, nature of the issues investigated, results both negative and positive, and the design criteria established. All processes developed, procedures completed, and lessons learned are described as part of each discussed issue. This report is partitioned into six sections and includes a list of acronyms (Appendix A). The contents of the report are as follows:

- Section 1 provides an introduction to the overall DAOC system, defining program goals and giving an overview of the document's organization.
- Section 2 discusses the DAOC services and the infrastructure into which the legacy applications were integrated. Design methodology for DAOC is also addressed.
- Section 3 presents a detailed description and discussion of the technical work completed for each of the four incremental phases of the DAOC system, including the specifics of each of the demonstrated software suites.
- Section 4 describes all technical issues investigated during each of the four incremental phases of the DAOC system, including assumptions, approach, applied processes, and resolutions.
- Section 5 summarizes the work done on DAOC and provides recommendations for future DAOC efforts.
- Appendix A presents the list of acronyms.
- Appendix B lists applicable and reference source documents.

- Appendix C is a background white paper, "Providing a Distributed Environment for the Airspace Deconfliction System (ADS)."
- Appendix D is a background white paper, "The Role of Distributed Planning in Tomorrow's C4I Environment."
- Appendix E consists of technical notes on SQL conversions for ADS-DAOC integration.
- Appendix F is an evaluation paper for DAOC Phase 3, "DAOC JFACC Jumpstart Hotwash Evaluation."
- Appendix G is the DAOC Interim Technical Report, which supplements the material in this paper.

2. SYSTEM INFRASTRUCTURE

2.1 Design Methodology

Because DAOC is primarily an integration project which has migrated over the course of the contract from the legacy application databases to a unified object model (for JFACC Jumpstart), both traditional (structured) and object-oriented methods have been employed as appropriate for each phase of the project. The methodology for detailed design was similar to preliminary design, the primary difference being the transition from higher levels of abstraction in the language and hardware requirements, to a language-specific detailed design.

CORBA supports an IDL (Interface Definition Language) that defines the object classes to be managed by the Object Request Broker (ORB) (for DAOC, this is Corbus or Orbix). The ORB vendor provides a tool to auto-generate code from the IDL, which the developer then fleshes out as necessary to provide user-defined operations. For the ORB services, documented IDL then constitutes the detail design.

Table 2-1 describes the detailed design methodology adapted for each programming language employed by DAOC. Occasionally, at the discretion of the DAOC software development manager, Program Design Language was generated for more complex units of the software design. PDL supports the standard control elements of structured programming, as well as those for object-oriented programming. PDL presents software design at a level of abstraction that facilitates review and validation and may be written as skeletal Ada or C++ code together with comments to describe the processing that takes place. The result is compilable but unexecutable skeletal code which later can be modified with the details.

2.2 DAOC Object Model Overview

For the requirements for which object-oriented techniques were appropriate, the software engineering methodology used was Object Modeling Technique (OMT), described in James Rumbaugh et al.'s *Object-Oriented Modeling and Design.* This methodology uses object-oriented terminology different from the traditional computer science; the DAOC SDP defines these terms. Our goal was simply to develop models of what the software must do:

- **Object Model.** This model describes the real-world object data types modeled in the system, and the static structural relationships between them.

Table 2-1. Definition of Detailed Design

Source Code	New/Modified	Detailed Design
C++	New	Documented class definitions
Ada	New	Documented package specifications
C	New	Documented header file with prototypes
Existing code	Modified	Identification of the modules that have been changed, and the nature of the changes

- **Dynamic Model.** This model shows control flow, concurrency, and temporal relationships between objects.

- **Functional Model.** This model describes data transformation effected by operations on objects, and the data dependencies between these operations.

For a more detailed description of the object modeling techniques employed by DAOC, refer to Appendix B of the Software Development Plan for the DAOC, LITG960033, 12 February 1996.

2.3 DAOC CORBA Services

DAOC uses CORBA services implemented through Corbus to achieve cross-WAN communications. The four CORBA services are the Reusable Database (RDB) Session Server, the Distributed Update Server (DUS), the CRIPMH, and the CDB API. Figure 2-1 shows the DAOC database access interfaces, the details of which are presented below.

2.3.1 The Reusable Database (RDB) Session Server

The RDB Session Server provides access by the application to the application database via the Corbus database managers. The RDB Session Server is implemented as a Corbus service, and provides database access across a WAN. It also provides a common interface to the Corbus database managers (e.g., Oracle or Sybase) which is transparent to the application.

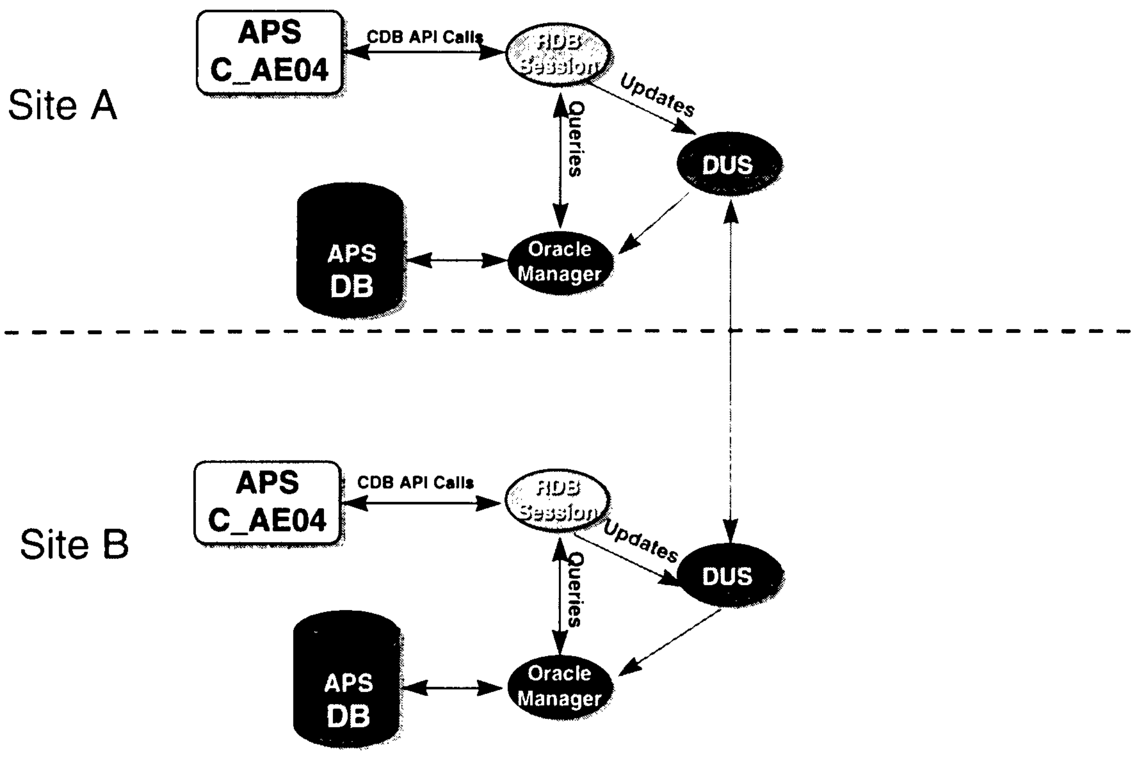

Figure 2-1. DAOC Database Access Interfaces

The RDB Session Server maintains a list of connections as Corbus objects in its object database. Each time a client requests a connection to the database controlled by the RDB Session Server, the Server compares the passed in usersname/password couplet against existing connection objects for a connection with matching usersname/password. If the connection exists and is available (defined below), the Server allows the client to use the Oracle or Sybase session object to connect to the database. Otherwise, the Server attempts to log in to the database on behalf of the client. If the login is successful, the Server returns the resulting Corbus session object to the client and creates a new object in its database containing the username/password combination and the unique identifier (UID) of the session object. When a client disconnects from the database, the Server merely marks the connection as being available. Clients may share the same connection, up to a maximum specified at Server startup.

The RDB also provides an interface with the DUS which is also transparent to the application. The RDB Session Server checks for a DUS for its database on initialization. If a DUS is not present, the RDB Session Server runs normally but with no replication of database changes.

An RDB Session Server is normally collocated with the database it controls (i.e., it is located on the same host), though it need not be. Because of a limitation in design, an RDB Session Server can control only one database. Since Corbus does not allow two managers of the same type to be started on the same host, it may occasionally be necessary to locate an RDB Session Server on a different host than the database it controls.

2.3.2 The Distributed Update Server (DUS)

The following provides a discussion of DUS capabilities and operating details.

2.3.2.1 DUS Overview. The Distributed Update Server (DUS) is the agent that provides database replication and synchronization functionality for the DAOC system, and thus is the cornerstone for distributed planning within the framework. The current version of the DUS is a CORBA-based service using Corbus. Rather than updating the local database directly, applications that need to provide distributed access to planning data invoke operations on the DUS running at the local site. The local DUS will then update the local relational database, and communicate with any number of predefined remote DUSs to perform the update on remote databases.

At each site, the DUS is responsible for verifying that local and remote updates are merged, and applied in chronological order. In the event of a communication failure, the updates are queued and retransmitted when communication is reestablished. Timestamping of updates helps prevent database inconsistencies from developing, even during long comm outages. If inconsistencies do develop during a comm outage, the DUS automatically detects them and notifies the user for conflict resolution. Using this approach, the replication of data to multiple databases is transparent to the application user. Moreover, data availability is maximized while consistency is still maintained.

Only minor source-code changes are required for applications to use the DUS rather than accessing the database directly. This transition is simplified by using a preprocessor that searches through Ada or C source code, and replaces database updates with calls to the DUS. Applications using the DUS make a straightforward operation call, and the DUS is then responsible for handling all communications with local and remote databases.

2.3.2.2 DUS Manager Overview. The DUS Manager is a complementary administration program that provides a Motif-based GUI to let a system administrator monitor the state of the DUS and its related database, and to customize the operation of the DUS. The current functionality allows: viewing a log file which shows all warnings and errors; delaying transmission of updates to a designated remote node; and isolating the local DUS node during massive critical database updates.

2.3.3 The CRIPMH

The CRIPMH is a Corbus-ized version of an APS-specific IPC package, which allows APS processes to communicate over a WAN. The original APS IPC package operates only over a LAN.

The purpose of the CRIPMH was primarily to allow operation of APS against a single central database, with the APS service instance package running only on the database server, and remote user instances communicating with it using the CRIPMH. After the central database architecture was abandoned in favor of the current DAOC architecture, the CRIPMH became unnecessary, since each site had a database server and its own service instance, and no interprocess communication among APS processes was necessary between sites. The intention for DAOC EP4 (which upgraded to the CTAPS 5.2 version of APS) was to re-integrate the original APS IPC package; however, time constraints and difficulty in generating an executable version of this module prevented this, and resulted in the CRIPMH's remaining in place.

2.3.4 The Common Database Application Programming Interface (CDB API)

CDB API provides client-side libraries for applications to interface with the RDB Session Server and the DUS. These interfaces are provided for C and Ada. In turn, C and Ada preprocessors provide translation of embedded SQL to the CDB API calls.

2.4 The Decision Support System (DSS)

The DSS is a Web-based client/server application that provides a view into the battlespace at the execution/engagement level. The DSS server performs distributed database operations currently on the APS and FLEX databases, but in theory is not limited to only these. Since the application is Web-based, it may be accessed via any platform with an Internet browser (e.g., Netscape) and access to the Internet. The status of the currently executing plan may be viewed in graphical format from the following perspectives:

- Objectives Explorer provides aggregated views of objective execution status.
- Mission Explorer shows which objectives are supported by individual missions.
- Unit, Mission, Resource, and other explorers give a more detailed representation of aggregated data.

Filters can be used in conjunction with any explorer to control the way data is aggregated.

3. SYSTEM OVERVIEW

The DAOC system is an integration project that has been designed and developed in an incremental software development effort to be a proof of concept of a system that ultimately allows planners at geographically separate sites to work collaboratively to create an ATO and monitor its execution. The four-phase design and development effort of the DAOC system, defined in Section 1.2, culminated in a technical demonstration of the software. The following sections describe each phase of the DAOC system as it was developed, and conclude with a discussion of the completed product for each phase, an Evolutionary Prototype (EP). Each section includes a purpose, requirements, design approach and rationale, special methods employed, and a report of the issues and concerns that surfaced from the associated JWID.

3.1 Phase 1 — Distributed FLEX

Phase 1 of the DAOC system provided the following capabilities:

- A single database at MCTSSA
- Remote processing at Hanscom AFB and USS Kittyhawk
- Changes to planning environment at remote databases notified by alert triggers
- Local mode executed during communication failures
- Provision of ongoing air operations picture

These capabilities were demonstrated at the DAOC JWID 95.

3.1.1 Performance Characteristics

DAOC operates in three possible modes: Master Site Mode, Remote Site Mode, and Local Mode. A DAOC site operating in Master Site Mode remains in this mode whether or not any remote sites are operational. A DAOC site operating in Remote Site Mode may reinitialize from local databases to switch to Local Mode. This switch would normally be motivated by failure of communications with the master site. A remote site that loses communications with the master site does not go into Local Mode automatically; only re-initializing from the local databases constitutes a change of mode. Finally, a DAOC site operating in Local Mode can reinitialize from the master site databases to switch to Remote Site Mode.

To summarize, the capabilities of each of the three modes of operation within the DAOC system are:

1. The DAOC master site is the central planning site, and serves as the data repository for the entire system. It also manages update propagation to operating remote sites.

2. A DAOC remote site is a remote planning site that uses the data stored at the master site. It is also responsible for registering with the master site to receive updates.

3. A DAOC site operating in Local Mode is a standalone backup to allow planning to continue on some level in the event of communication failure between a remote site and the master site. For Phase 2, this mode of operation does not include resynchronization of data with the master site.

3.1.2 Design Rationale and Approach

3.1.2.1 Phase 1 DAOC Interfaces. The Phase 1 DAOC Interfaces for the JWID 95 are shown in Figure 3-1.

3.1.2.2 JWID 95. The DAOC JWID 95 configuration is shown in Figure 3-2.

The JWID 95 scenario comprised the following locations:

- MCTSSA, Camp Pendleton, California
- USS Kittyhawk, Pacific Ocean
- Ft. Franklin, Hanscom AFB, Massachusetts

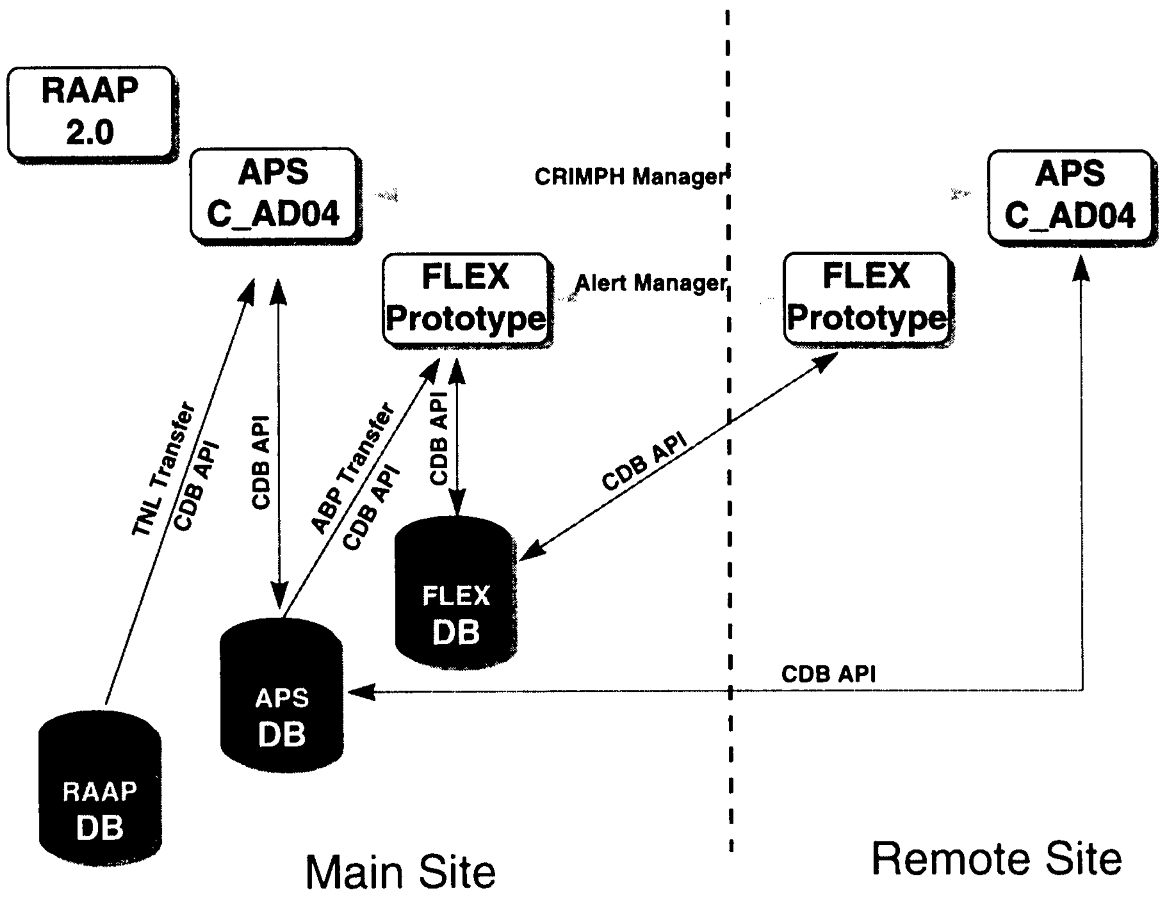

Figure 3-1. JWID 95 Interfaces

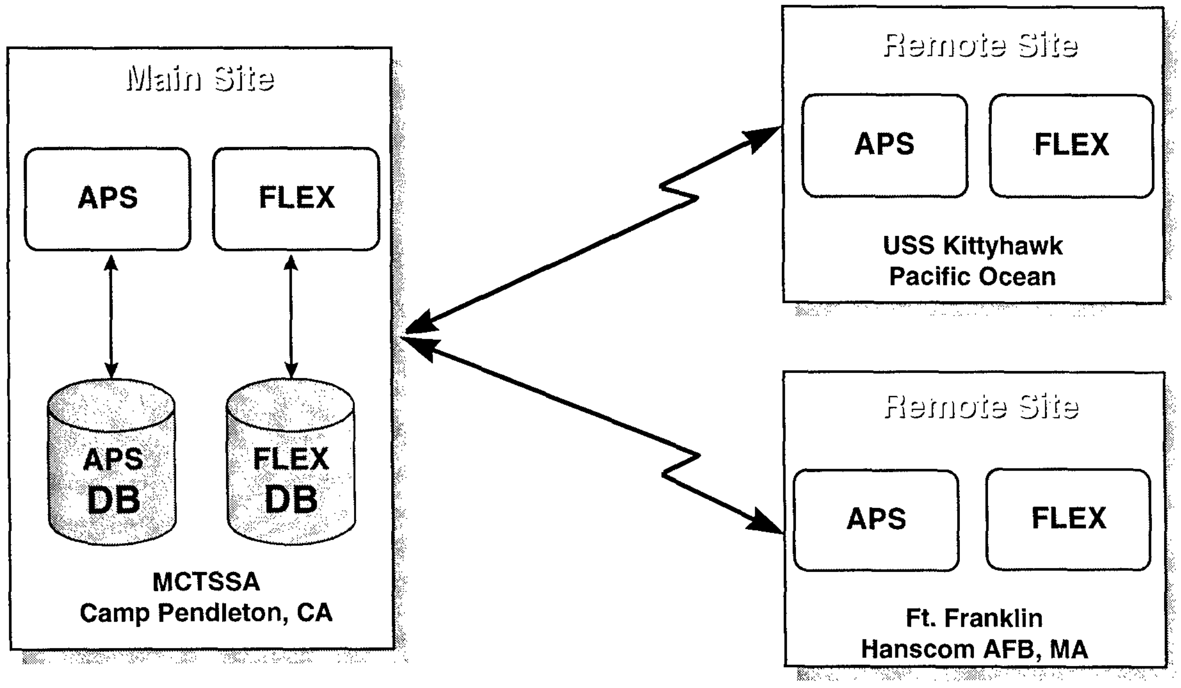

Figure 3-2. DAOC JWID 95 Configuration

3.1.3 Issues and Resolutions

The primary issue encountered at JWID 95 was the frequent loss of network communications, particularly with the USS Kittyhawk, and the resulting loss of planning ability at remote sites which were cut off from the central database site. During JWID 95, this issue was addressed by allowing the system to switch from the central database to a local copy of the database. This allowed system operability at all times; however, since there was no mechanism to synchronize these local databases with the central database, all work done on the local databases was lost when connectivity was regained and the system went back into distributed mode. This experience was the primary driver for the subsequent architecture of the DAOC system.

3.2 Phase 2 — Distributed FLEX and APS

The DAOC Phase 2 demonstration system integrated existing AOC applications into a distributed planning suite. Applications included were Rapid Application of Air Power (RAAP), Advanced Planning System (APS), Force Level Execution System (FLEX), and Local Attack Controller (LAC). The DAOC suite also provided an interface between the JTF Planner and the AOC via the Master Air Attack Plan (MAAP) produced by the ACPT.

Phase 2 of the DAOC system provided the following capabilities:

- Application databases employed at every site
- Databases synchronized by Distributed Update Server (DUS)
- Reachback to Rome Lab for Target Nomination Transfer (TNL) from RAAP

- Coalition planning with English-speaking allies
- A more robust architecture with respect to communication slowdown or failure

These capabilities were demonstrated at the DAOC JWID 96.

3.2.1 System Performance

3.2.1.1 Performance Characteristics. The JWID 96 architecture for DAOC, which became the final architecture for this prototype system, attempted to address the problems observed at JWID 95 by migrating from a strict client/server system (which carries with it the assumption that if the server is inaccessible, the client will be inoperable) to a server/server system, in which each DAOC site has equality with all others and is capable of being fully operational even if isolated from the network. Since much of JWID 95 was spent operating from local databases, the decision was made to make the local database an integral part of the system. This introduced a new consideration: keeping the data storage consistent across systems. The JWID 95 architecture distributed the integrated applications by providing CORBA services to allow access by the applications to their databases across the WAN. These services were generally application-specific, the requirements for them being tightly coupled with the application's internal interface structure. For JWID 96, the applications were self-contained at each site. The need for the specialized communication services was largely negated (except for certain data-transfer functions which needed cross-WAN capability, e.g., the TNL pull from RAAP to APS). The only service that operated over the WAN was the Distributed Update Server (DUS), the purpose of which was to capture database changes as they were made (at the transaction level) and immediately ship them to all other copies of the same application database. The DUS was designed to handle communication outages by containing to try to put the changes through until it was successful.

3.2.1.2 Baseline. The baseline AOC suite from JWID 95 was used as the starting point for the DAOC system. The CTAPS version 5.2 AOC suite was integrated into the DAOC system at a later time, including RAAP, APS, FLEX, and ADS.

3.2.2 Distributed FLEX and APS Architecture

3.2.2.1 Phase 2 DAOC Interfaces. The Phase 2 DAOC Interfaces for the JWID 96 are shown in Figure 3-3.

3.2.2.2 JWID 96. The DAOC JWID 96 configuration is shown in Figure 3-4.

The JWID 96 scenario comprised the following locations:

- Ft. Bragg, North Carolina
- Shaw AFB, South Carolina
- Defence Research Agency (DRA) Malvern, United Kingdom

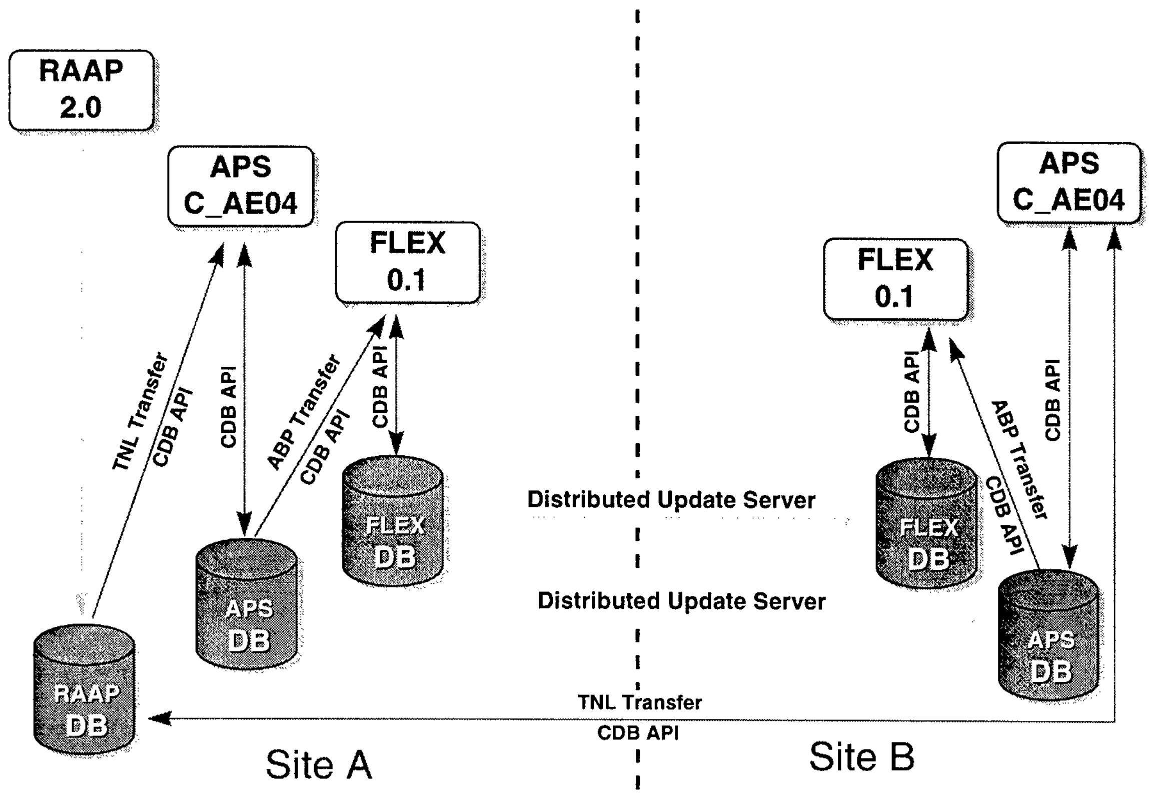

Figure 3-3. JWID 96 Interfaces

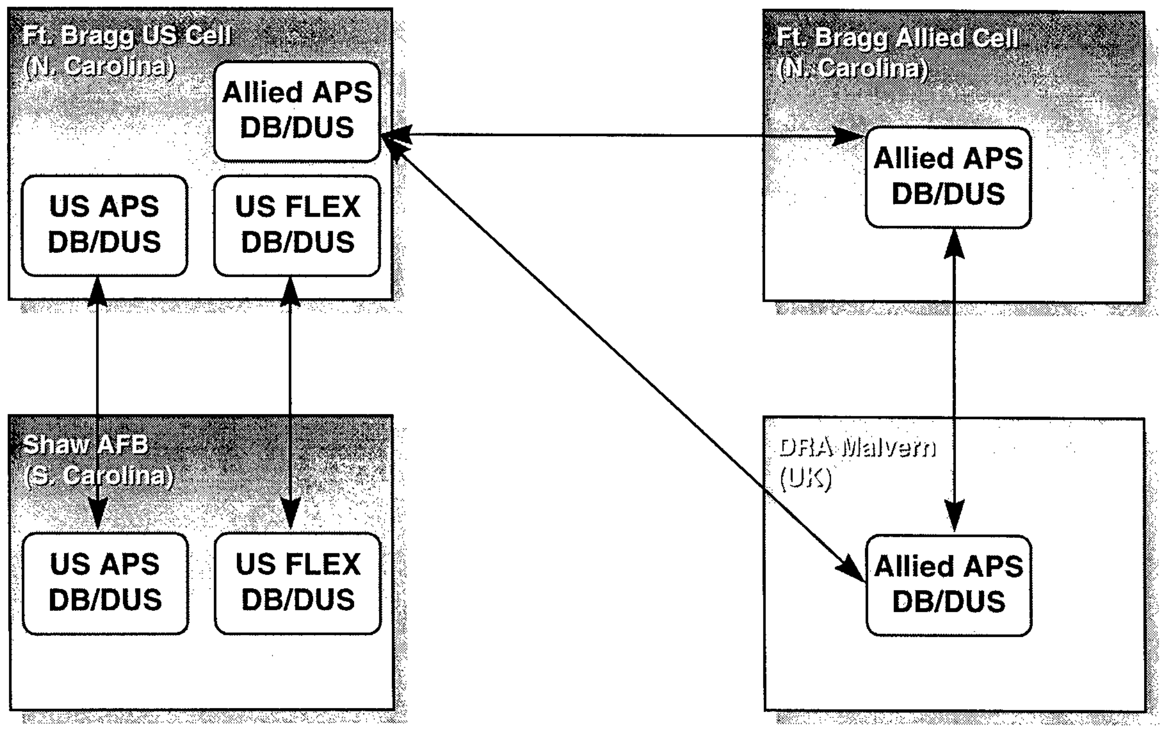

Figure 3-4. DAOC JWID 96 Configuration

3.2.3 *Issues and Resolutions*

3.2.3.1 DAOC. To effect the RAAP-to-APS interface over the WAN (involving the export of the Corbus Sybase manager from the RAAP machine and its import by the APS machines) a couple of unanticipated steps had to be performed. In addition to the import/export of sybase and RDBSession, the SybaseSession manager (sybses, type 62) had to be defined as a service on both the RAAP and APS machines, and added to the RAAP machine as a non-autostarted service. The service is never actually started, and will always show as a "not active" installed service on the RAAP machine. The SybaseSession manager then had to be exported from the RAAP machine and imported by the APS machines. Also, the symbolic database name used by the DAOC services (in this case, RAAPDBRL) had to be linked to the actual name "theater" by a tropic operation on the Sybase manager so that the TNL data tables would be visible. Once these steps had been performed, the TNL pull succeeded over the WAN.

Another problem that occurred several times was the Corbus kernel's running out of process descriptors. Symptoms of this are: (1) doing "cr showk /serv" results in "kernel not running or in wrong configuration," but the kernel process is actually still running; (2) the kernel log contains "out of process descriptor" messages; (3) in many cases, Oracle worker managers are not cleaned up, so there are many workers with no corresponding sessions. The Corbus kernel defaults to a process limit of 40, which is easily reached by the DAOC services. There is a command line option to crboot that allows the limit to be raised (/maxproc=n, where n is the new limit.) For this JWID, n was set to 200, which proved adequate.

Running out of tablespace in the APS database became an issue after several days of ATO generation. This resulted in a failure of the APS clone operation which was not reported to the user; in other words, the clone operation reported success but the resulting ATO was unusable. Symptoms of this are: (1) many ATOs in the database (in this case, more than 10); (2) apparently successful clone operation in which, however, the newly cloned ATO cannot be initialized in planning mode; (3) the cloning log file (which has a unique name based on process id and is best found by "ls -lrt" in the directory which contains the APS log files (usually /usr/tmp)) contains many Oracle error messages such as "out of table space" and "cannot create table xxx"; (4) if XX1 is the source ATO, and YY1 is the newly cloned ATO, then "select * from cat" performed on XX1 in sqlplus will return many more rows than when performed on YY1. The solution, obviously, is to delete some ATOs and/or increase the tablespace in the APS database.

It is interesting to note that the above two problems (running out of tablespace in Oracle and running out of process descriptors in Corbus) occurred simultaneously at this JWID, and that cleaning up the database at first appeared to eliminate the Corbus problems. Our conjecture is that when the tablespace limit is approached, database transactions take longer and therefore the Corbus worker managers and the Oracle session processes stay open longer, providing the opportunity for more processes to build up and for the process limit to be reached.

A problem with the DUS was detected and corrected during the course of JWID. The bug related to preserving the order of database updates following a communications failure. Communications were lost to the UK site during a time when planning activities were still happening at the Ft. Bragg allied site; when communications were restored, several escort missions that had been created during the outage period failed to be propagated to the United Kingdom database. The reason was that the DUS had pushed the escort missions through before the escortee missions had been created. This was because of a logical error with the DUS's retry mechanism: it would continue traversing the queue of pending updates after a particular update failed to get through, so that the order in which the updates would be applied upon restoration of communications was effectively random. This was corrected to keep trying the first update that failed until it was successfully propagated. The fixed version of the DUS was thoroughly tested using APS during the last week of JWID and appeared to be good.

A problem with the Corbus kernel was observed twice in the situation where communications failed while planning activities were taking place at the Ft. Bragg allied site. The cause is not yet thoroughly understood. In the first case, communications failed between Ft. Bragg and the UK while the Ft. Bragg site was in use. There was no activity on the UK side. The Corbus kernel appears to have been caught in the middle of an operation, and continued to try to contact the UK site with a 60-second timeout. During the timeout period, all Corbus processes blocked, causing the application to be unacceptably slow. This behavior of the kernel remained unchanged regardless of what mode the DUS was in (non-propagating, local), so the conclusion is that the trouble is not because of the activities of the DUS. The problem was observed again when a power failure occurred in the Ft. Bragg U.S. cell while planning was being done in the Allied cell. In this case, the timeout was 19 seconds (presumably because both sites were on the same LAN), but otherwise the behavior was the same. Note that in the second case, both sites resided in the same Corbus cluster. The kernel log files from these incidents were preserved and passed on to Mike Dean of BBN, who has promised to look into the problem.

The DAOC modifications to APS generated some user frustration, particularly the APS Service Instance (called ALGS) reinitialization issue. The users found it difficult to understand and cumbersome. The next version of DAOC APS should be changed so that ALGS updates itself from database changes, probably through the use of database triggers in the manner of the FLEX knowledge base. If this is accomplished, it will also obviate the need for site IDs, which did not generate much negative reaction only because the users were protected from having to deal with them. Also, the need to clone an ATO separately at each site tended to be a roadblock. Some thought should be given to either distributing the OraPerl scripts or (probably a better solution) distributing the call to the OraPerl scripts.

The TNL pull interface is not user-friendly for operators who are unfamiliar with the system. It would be a good idea to change it so that the usernames/passwords for the APS and RAAP databases are provided by environment variables (since they will generally remain constant for the duration of a demo), rather than requiring the user to input them.

The DAOC demo, because of the nature of the project, is not particularly visually appealing. In the future, it may be worth considering sprucing up the DUS GUI to show more of what is happening internally, e.g., showing graphically which connections are active.

3.2.3.2 Data and Legacy Systems. One of the major roadblocks at the beginning of JWID 96 was the lack of adequate setup data. The JA1 ATO which was provided to us did not contain assets that were appropriate, especially for the Allies. Several days were spent during the first technical free-play week just making the baseline theater data useful, and during the course of JWID it became necessary to add additional theater data. Also, no RAAP data was provided at all, so the TNL had to be created by hand at Rome (the home of the RAAP machine) from a preliminary copy of the MAAP. This was made more of an issue because of the security partition between the U.S. and Allied sides; whenever setup data was changed on one side, the other side had to be brought up to date either by database export/import or by duplicating the changes in the application. Most of the time this synchronization was accomplished by database export/import because of the high probability of human error inherent in the other method.

The ABP Transfer (ABPT) package has a number of problems, including:

1. The only way to discover which FLEX database constraints need to be disabled is to run ABPT until it fails. Also (this is more a problem with the FLEX database) the constraint names vary between instances of the FLEX database, so discovering constraint names to disable on one machine gives no reliable information about which to disable on another.

2. The ABPT in its original form assumes that the FLEX and APS databases will run within the same Oracle instance. The code had to be modified for DAOC to remove that assumption.

3. When ABPT was first used to import a JWID 96 ATO into FLEX, the following problem was observed regarding bases previously unknown in the FLEX database: The ABPT proceeds with no errors, but when the FLEX knowledge base initializes, the new bases are not read in. The reason for this was that the FLEX KB checks the country code(s) of units assigned to a base, and then checks that these are not enemy countries before adding the base to its list. In the case of the JA1 data, units did not have assigned country codes. This had to be corrected in APS and the new country codes added to TH_COUNTRY_TAB in the FLEX database before the KB would recognize the new bases.

A few issues arose with APS (version C_AE_04) which are documented here for future reference:

1. Under certain circumstances, cloning an ATO in APS results in wiping out the takeoff and landing times of missions in the APS database. APS itself operates properly when this happens, but when ABPT is performed to pull the ATO into the FLEX database, the missions have no takeoff/landing times in FLEX; this creates problems with the FLEX marquee and the alerts. When an ATO has planned missions and is cloned with the missions (attack or support missions), the clone script checks for TOT/TFTs in the APS database and only copies the takeoff and landing times if TOT/TFT is filled in. Since many missions legitimately do not have TOT/TFTs, this results in many missions with no takeoff and landing times. It can be circumvented by either deleting and recreating the missions in APS (which restores the takeoff and landing times in the database) or by

filling in the TOT/TFT fields which are null in the original ATO before the clone is performed. At JWID, this did not have much of an impact, since the clones were generally made without planned missions.

2. When the Oracle tablespace was full, the APS clone operation failed but still reported success. This is discussed in greater detail in Section 3.2.3.1.

3. Occasionally, and apparently randomly, escort requests were unable to be filled. The popup error message and the error log say only "unable to create mission" with no further explanation. The reason for this is unknown.

4. APS is very sensitive to changes in ATO setup data when cloning an ATO. Two instances of this are known: (a) failure to fill in sunrise and sunset times will not generate an error at creation time, but will prevent ALGS from initializing; also, the only way to add them after ATO creation time is to create another clone; (b) adding a tasking agency to the ATO setup data seems to introduce another layer of filtering for available resources, and causes no available resources to show on the planning worksheets.

FLEX was used at JWID 96 only for ATO monitoring, not change generation; therefore, not too many FLEX issues came up. However, the following did attract some attention:

1. Occasionally the FLEX database would be updated but the marquee would not be. This may be an artifact of running under Solaris 2.4, since this problem was not observed on the copy of DAOC FLEX that runs under Solaris 2.3.

2. The sunrise/sunset bar on the marquee does not match the sunrise/sunset data in the APS database for the ABP being monitored.

3.2.3.3 Personnel. One major problem at this demo was that, counter to expectations, none of the APS operators had previous experience with the application. The technical support and coordination personnel from Logicon and Rome Lab were in the position of having to provide APS training and support to new users, which we were not adequately prepared to do.

3.3 Phase 3 — JFACC Jumpstart

Phase 3 of the DAOC system provides the following capabilities.

3.3.1 Plan Generation

- Provides mission planning and execution for air operations
- Provides Web-based Decision Support System (DSS)
- Integrates JTF ATD servers into the DAOC planning environment

3.3.2 Decision Support System

- Provides a Web-based visualization system in graphical format

- Provides logistical and statistical information

3.3.3 JFACC Jumpstart Architecture

Figure 3-5 depicts the layout of the planning elements that compose the DAOC architecture for the JFACC Jumpstart execution.

3.3.4 JFACC Jumpstart Integration/Demonstration

The JFACC Jumpstart integration and demonstration effort has been previously detailed in the interim DAOC technical report. For specifics involving this effort, refer to that report, included as Appendix G of this FTR.

3.3.5 Issues and Resolutions

The DAOC JFACC Jumpstart capability was analyzed and evaluated immediately after the integration/demonstration effort. For specifics, refer to Appendix F of this FTR, "DAOC JFACC Jumpstart Hotwash Evaluation."

3.4 Phase 4 — Blue Flag 97 DAOC Capability

The DAOC capability provided for Blue Flag 97 built upon the DAOC JWID 97 capability by integrating the latest CTAPS (version 5.2) versions of APS and ADS, and version 2.4 of FLEX.

3.4.1 Performance Characteristics

Performance of the DAOC framework did not change for the DAOC Phase 4 capability; however, the integrated CTAPS applications contained minor operating changes as a result of their upgrade.

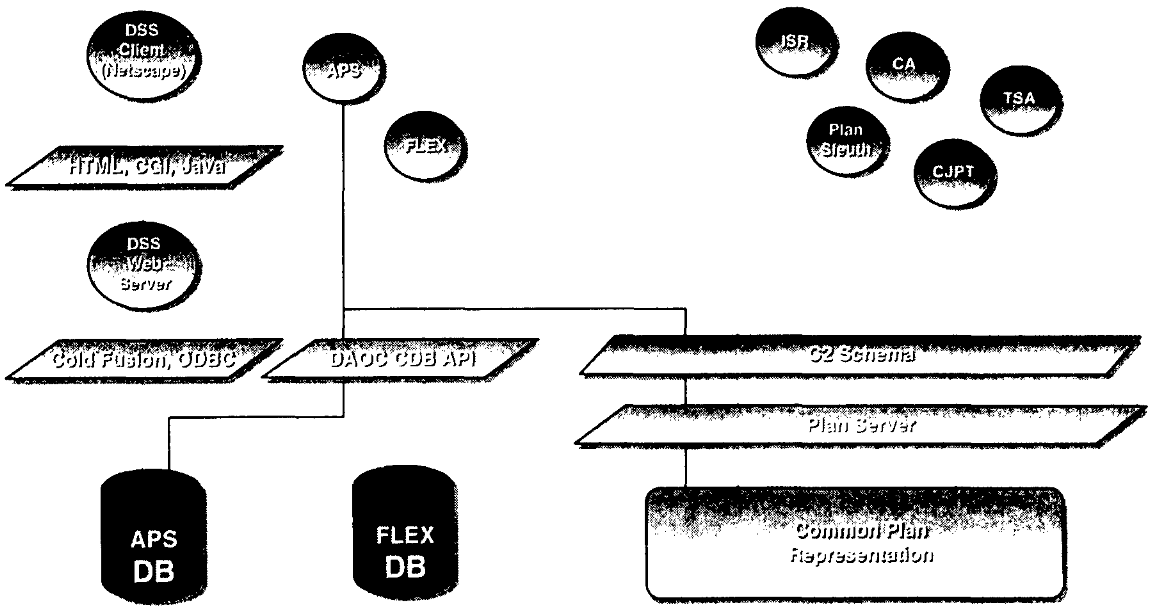

Figure 3-5. DAOC/JFACC Planning Architecture

3.4.2 Phase 4 DAOC Interfaces

Phase 4 interfaces for the DAOC JWID 97 implementation are depicted in Figure 3-6.

3.4.2.1 JWID 97. The DAOC JWID 97 configuration is shown in Figure 3-7.

3.4.3 Summary

The JWID 97 demonstrated the utility of distributed, collaborative air mission planning and execution monitoring (including a representative sample of logistics and reachback data) within a U.S./coalition structure. It used coalition communication networks as well as U.S. networks with the appropriate multilevel security (MLS) devices and databases to allow real-time collaboration between the coalition air operations cell and each member's national air operations cell located back in the host country. The demonstration illustrated that the JFACC, while airborne during transition from JFACC (rear) to JFACC (forward), could maintain situational awareness of the theater during flight. DAOC served as the foundation of this effort, providing air campaign planning/Air Tasking Order (ATO) generation functions that could operate in a distributed manner with provisions for sharing data at different security levels.

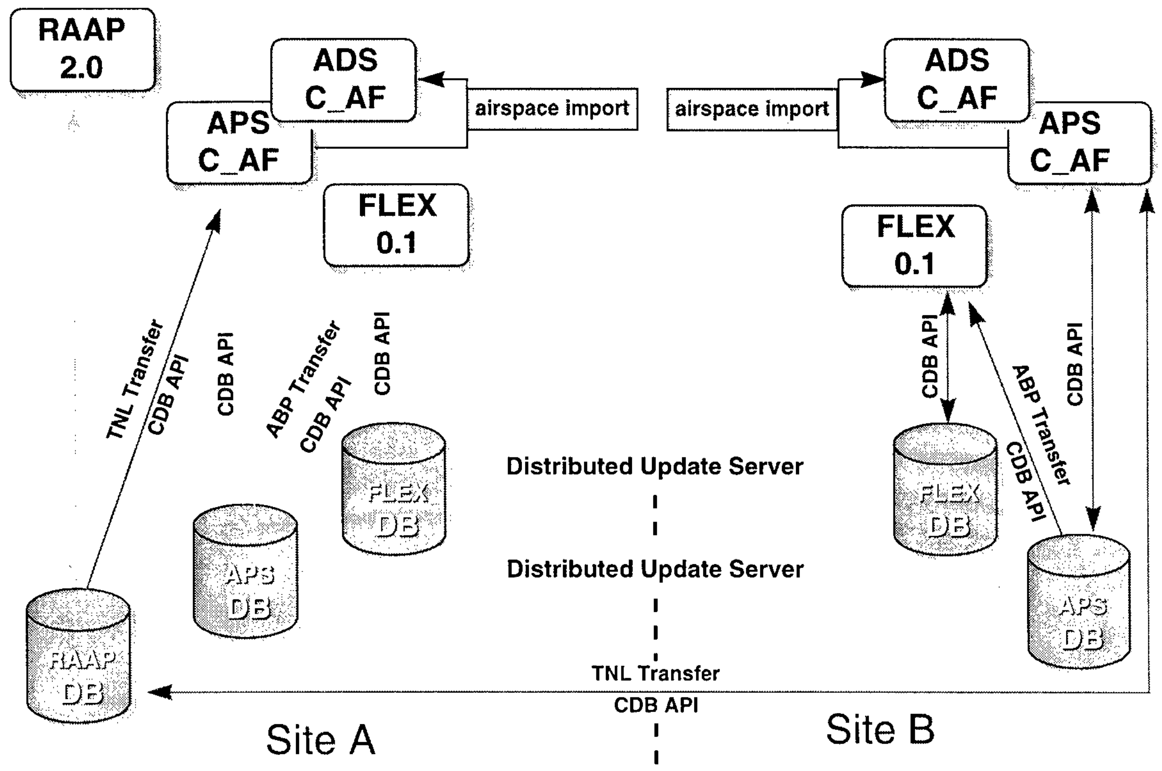

Figure 3-6. DAOC JWID 97 Interfaces

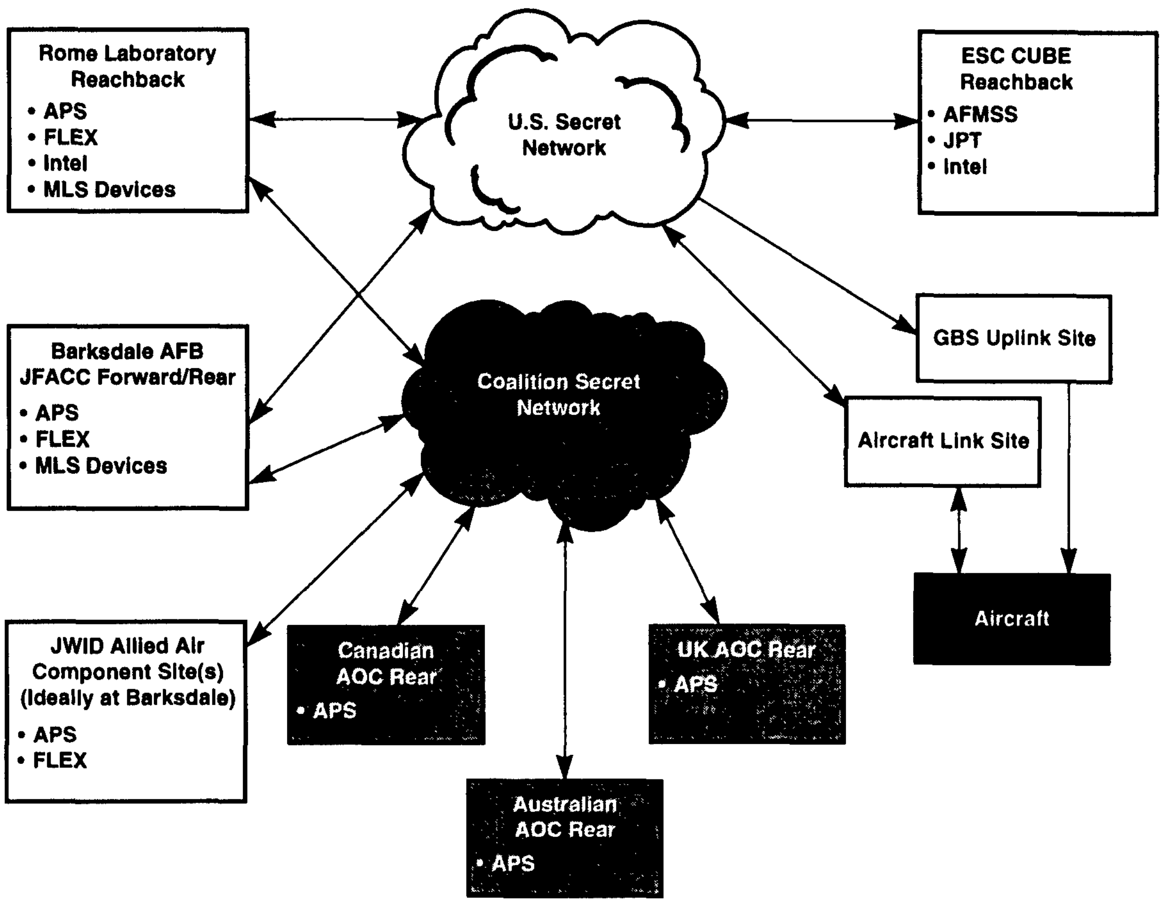

Figure 3-7. DAOC JWID 97 Configuration

3.4.4 Implementation Details

The DAOC JWID 97 scenario comprised the following sites:

- Barksdale AFB
- Hanscom AFB
- Winnepeg, Canada
- Rome Laboratory
- USAF Speckled Trout (airborne)
- USS Stennis (at sea)

The hardware and software consisted of:

- Sun SPARCstation 10 and 20s
- PowerLite (SPARC laptop)
- Sun OS 4.1.4

- Sun Solaris 2.5
- Advanced Planning System (APS) CTAPS Version 5.1.3
- Force Level Execution System (FLEX) version 2.4
- Airspace Deconfliction System (ADS) CTAPS version 5.1.3
- Decision Support System (DSS) version 1.2
- Rapid Application of Air Power (RAAP) version 2.0
- Combat Intelligence System (CIS) version 1.2
- Collaborative Virtual Workspace version 2.7
- Global Command and Control System version 2.0

Although not depicted in Figure 3-7, DSS capability was available at each node at each site. DSS is a Web-based application, and as such requires only an Internet browser (e.g., Netscape) to be accessed. The DSS server resided at the Barksdale Rear position.

3.5 DAOC Advanced Technology Demonstration (ATD)

The DAOC ATD was presented on September 8, 1997. The ATD configuration is depicted in Figure 3-8.

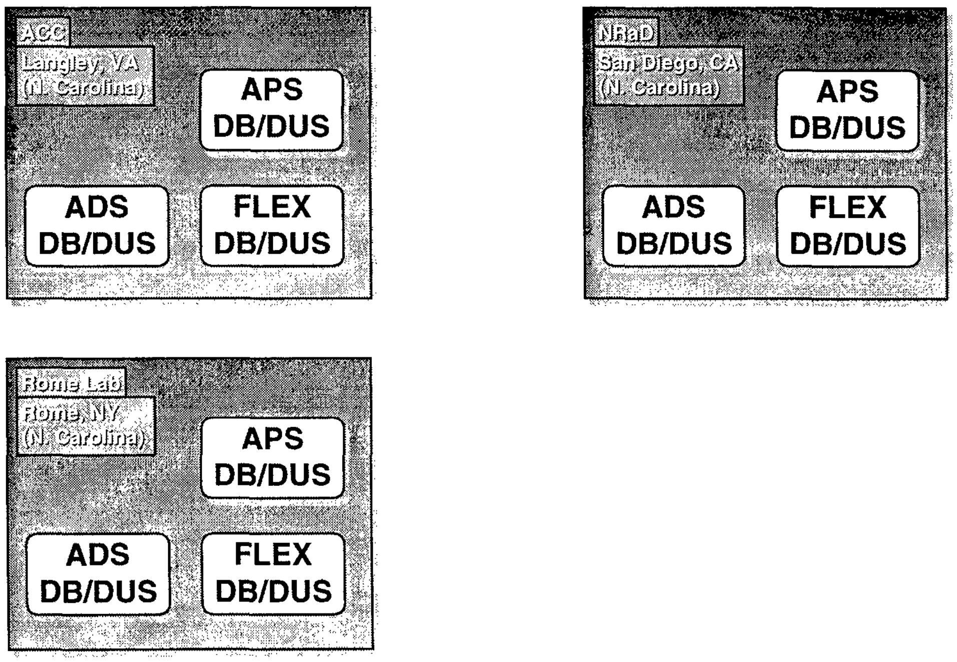

Figure 3-8. DAOC ATD Configuration

In order to maintain clarity, connectivity is not depicted in the above diagram; however, each application node at each geographic site had connectivity to its counterpart, e.g., distributed collaboration was possible between all APS nodes. In addition, Collaborative Virtual Workspace (CVW) was used during the demonstration for visual and audio communications between sites. A pre-scripted scenario was enacted which demonstrated the distributed collaborative capabilities provided to the legacy CTAPS applications as a result of their integration into the DAOC framework. Survivability and recovery of communications was demonstrated by simulating a comm outage between the NRaD and the ACC sites, and showing that FLEX modifications made at NRaD appeared at ACC following restoration of communications.

4. TECHNICAL DISCUSSION

The DAOC system is an evolving prototype. It has been updated to examine uncertain or critical areas identified before or during each phase of the software design. This section documents the approach, rationale, and methods employed to resolve all issues that ensued.

GFI software has been used in the delivered DAOC system. The baseline AOC suite from JWID 95 has been used as a takeoff point for the second phase of the DAOC system. The CTAPS version 5.2 AOC suite has been integrated into the DAOC system, including RAAP, APS, FLEX, and LAC. Corbus ORB was also used.

4.1 Technical Issues: JWID 95

The first phase of the DAOC evolutionary prototype development met the issues of how to maintain operation during communication failure or slowdown, and how to perform an initial synchronization of object repositories. For details on the DAOC Phase 1 effort, refer to Appendix G of this FTR, the DAOC Interim Technical Report.

4.2 Technical Issues: JWID 96

Several technical issues emerged during the development of the DAOC system prototypes that required investigation. What follows is a description of the following issues, and the analyses and tradeoffs they initiated:

- Data synchronization
- Joint coalition planning and execution
- Partitioning of data into areas of responsibility (AORs)
- Most robust architecture with respect to communication failure

4.2.1 Data Synchronization

Data synchronization for a distributed system concerns the consistency of the data as updates are performed across a system. The use of a CORBA-compliant distributed system ensures that data at geographically separate locations is kept in sync and provides operability across a network system, both LAN and WAN. The following section details our rationale and approach for selecting and ultimately implementing a peculiar CORBA-based system, the Distributed Update Server (DUS), in the development of the DAOC system.

4.2.1.1 Introduction to the Distributed Update Server (DUS). The DUS was developed to provide distributed planning capabilities for the DAOC program. The current version of the DUS is a CORBA-based service using Corbus by Bolt, Beranek and Newman (BBN).

In general, the DUS provides for the following capabilities:

- ***CORBA.*** The DUS uses CORBA to produce a distributed planning environment, and provides operability across a WAN or LAN. Application databases located at geographically separate sites are kept in sync.

- ***Data Availability.*** Data availability is emphasized so that planning done at remote sites is available in near real time. Applications can operate normally through communication failures, and a retry capability ensures that the data is not lost during communication outages.

- ***Data Consistency.*** Data consistency is maximized through a timestamping mechanism that ensures that updates are applied in the correct order.

- ***Integration.*** Automated tools allow easy integration with applications.

4.2.1.2 Rationale Behind the DUS. By using the DUS approach, replication of data to multiple databases is transparent to the application user. Moreover, data availability is maximized while consistency is still maintained.

Only minor source code-changes are required for applications to use the DUS rather than accessing the database directly. This transition is simplified by using a preprocessor that searches through Ada or C source code, and replaces database updates with calls to the DUS.

Rather than updating a local database directly, applications that need to provide distributed access to planning data invoke operations on the DUS running at the local site. The local DUS will then update the local relational database, and communicate with any number of predefined remote DUSs to perform the update on other remote databases.

At each operational site, the DUS is responsible for verifying that local and remote updates are merged and applied in chronological order. In the event of a communication failure, the updates are queued and retransmitted when communication is reestablished. Timestamping of updates helps prevent database inconsistencies from developing, even during long communication outages. If inconsistencies do develop during a communication outage, the DUS automatically detects them and notifies the user for conflict resolution

4.2.1.3 Underlying Mechanics of the DUS. The DUS is a multi-threaded, persistent (always running) CORBA service. To transparently replicate data, it maintains three types of queues:

1. ***Update Queue.*** An update queue is used to apply local commits and remote updates in chronological order. Local commits are timestamped before being inserted into this queue. Remote updates are timestamped at the point of origin. All entries in the update queue are applied in chronological order.

2. ***Transmit Queue.*** A transmit queue is used to buffer local updates so that they are reliably transmitted to remote sites. The transmits are done asynchronously by invoking a one-way operation on each remote DUS. Periodically, the local DUS checks for the receipt of an acknowledgment. The retransmit is attempted until an acknowledgment is received. There is one transmit queue for each remote site.

3. ***Results Queue.*** A results queue is used to buffer status codes from each remote DUS after the update has been applied at the remote site. If the results from all the remote sites and from the local update are not the same, then a database inconsistency has developed, and notification is given at both sites. There is one transmit queue for each remote site.

Applications using the DUS make a straightforward operation call, and the DUS is then responsible for handling all communication with local and remote databases.

4.2.1.4 Life Cycle of a DUS Transaction. Figure 4-1 depicts the processing of a typical application transaction as it travels through the DAOC system via the DUS architecture.

4.2.1.5 DUS Summary. In summary, implementation of the DUS achieved the following:

- Allowed a completely distributed environment regardless of geographical separation
- Resynchronized databases when communication resumes after a communication failure in the application
- Minimized code changes to applications to which it interfaces
- Made replication of data transparent to the application user
- Operated over a WAN between sites separated by as much as 400 miles (demonstrated at JWID 96)

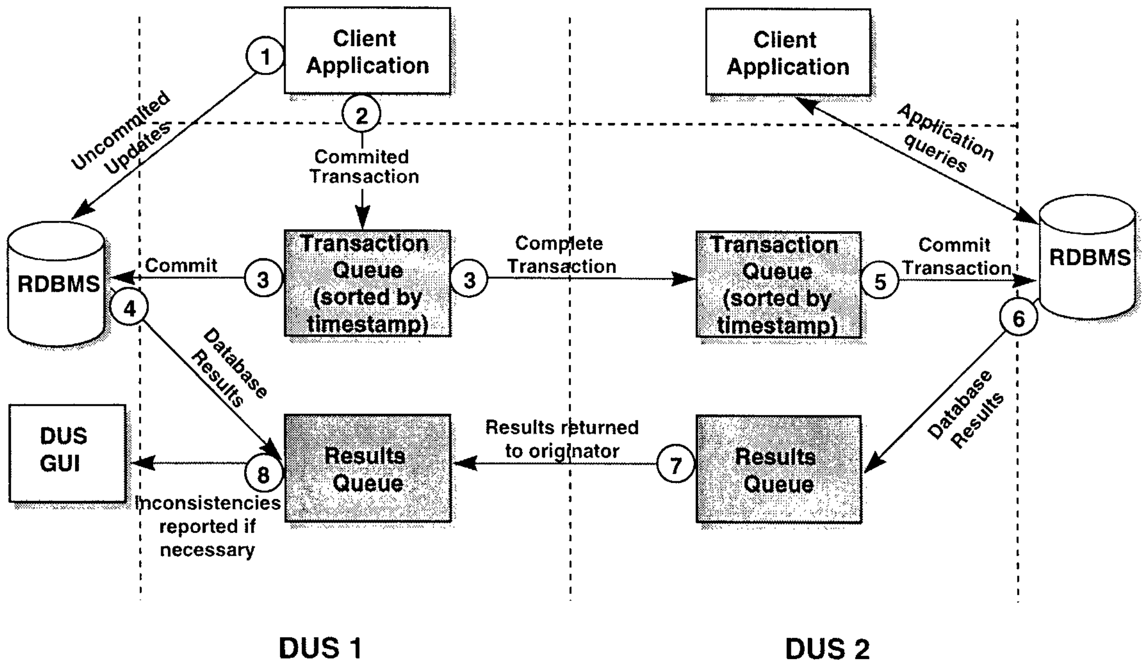

Figure 4-1. Life Cycle of a DUS Transaction

- Provided consistency of data even following communication outages (robustness through communication failures was demonstrated at JWID 96)

4.2.2 Joint Coalition Planning and Execution

For JWID 96, DAOC had two complete systems running on disjoint LANs, one at the U.S. Secret level and one at the JWID96CO level, which was equivalent to Secret Releasable AUSCANZUKUS. The two LANs were linked electronically through the NightHawk trusted guard, which allowed data transfers from one security level to the other, subject to human review. Any data not human-readable was not able to be passed by the guard. Near the end of the ATO planning process, the ATO planned on the JWID96CO side (consisting of missions planned by Australian, UK, and Canadian planners) was passed across the guard and manually folded into the ATO produced by the U.S. planners. The result was an integrated coalition ATO. The method was cumbersome and subject to human-introduced error. During the JWID 97 effort, some investigation was done into replacing the NightHawk guard with a trusted Oracle bridge machine, which would allow the DUS to operate across security levels using stored procedures to replace the human review process. While this capability was not implemented for JWID 97, it seems a viable alternative and is worthy of further investigation.

4.2.3 Partitioning of Data into Areas of Responsibility (AORs)

Since the DUS architecture emphasizes data availability while attempting to maintain data consistency, data inconsistency can be compromised if multiple sites are operating on the same data at the same time (e.g., two DAOC APS operators modifying the same mission at the same time). This issue was handled for the DAOC prototype by the conops of assigning each DAOC site an area of responsibility. This conops is not inconsistent with actual practice; in the case of JWID 96, planners from each of the coalition nations took responsibility for their nation's assets. In addition, commander's guidance provides specific roles for each nation's assets (e.g., Australia will provide fighter escort for American strike missions in a given region). During the course of this and other JWIDs, this conops approach proved adequate. However, the issue of data contention should be addressed in future DAOC work.

4.2.4 Handling of Communication Failures

Based on the lessons learned from the JWID 95 central database architecture, the JWID 96 architecture had several built-in capabilities that allowed operation through, and recovery from, communication failures. A full application database at every site provided the capability to continue work even if there was no WAN connectivity to any other site. The retry mechanism of the DUS provided the ability to recover from communication failures in a manner nonintrusive to the user. When communications resume, a DAOC site sends and receives changes that were made during the outage period. This capability was thoroughly tested during JWID 96, JFACC Jumpstart, and JWID 97.

4.3 Integration of AOC Applications into DAOC

The DAOC system endeavored to integrate existing AOC applications into a framework which would allow for a distributed collaborative environment. Initially, FLEX was chosen as the first such application for integration. APS followed, and an APS/FLEX capability was demonstrated at JWID 96. The Airspace Deconfliction System (ADS) was added to the list of integrated applications for JWID 97. This same capability was demonstrated at the DAOC ATD in

September 1997. The existing capability was enhanced for Blue Flag 97 by upgrading the integrated applications to CTAPS version 5.2.

The DAOC interfaces to the AOC applications are provided transparently by the DUS. Figure 4-2 depicts the DAOC system in its JWID 96 configuration; however, the basic paradigm has remained constant through subsequent integrations.

In general, integration of existing AOC applications into the DAOC framework required minimal source-code changes to the application. To convert embedded SQL code in the applications, C and Ada preprocessors were developed. These preprocessors would automatically replace the SQL with CDB API library calls, thus preserving original functionality and, in addition, creating an interface to the DUS to utilize its distributed database management capabilities.

4.3.1 APS Integrated into DAOC

Three versions of APS were integrated at different times into the DAOC system: version CAD-04 was used at JWID 95; version CAE-04 was used at JWID 96 and 97; and version CAF-06 (representing an upgrade to CTAPS 5.2) was used for Blue Flag 98-1.

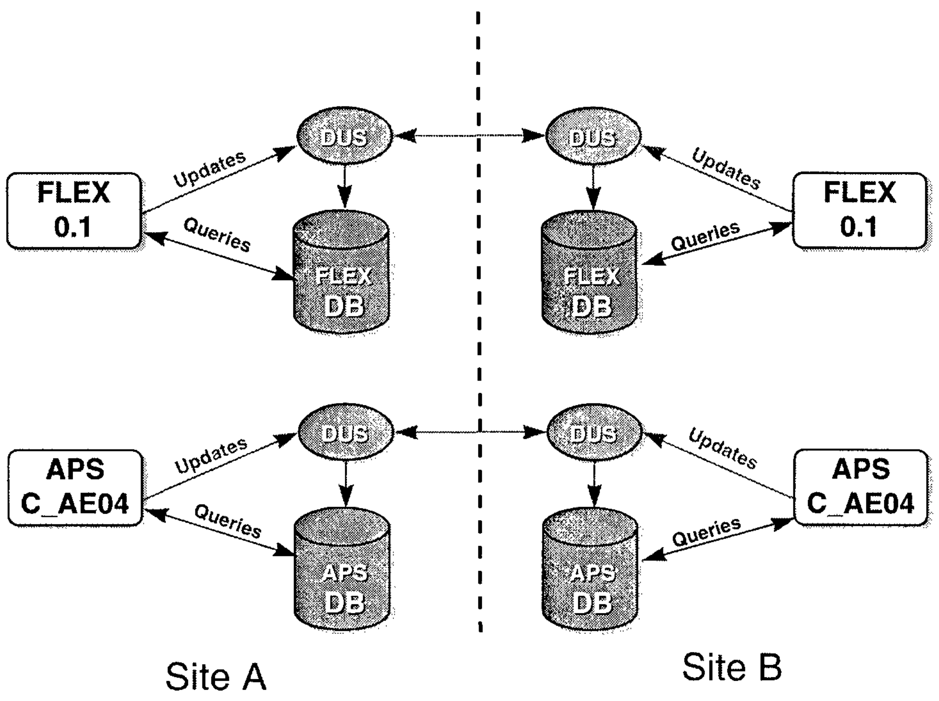

Figure 4-2. JWID 96 Configuration of DAOC System

Because APS is designed as a locally distributed system, some issues arose when integrating it with the wide-area distribution provided by the DAOC suite. In particular, the architecture of APS is based on the assumption that there is at most one copy of the APS Service Instance for each ATO in the database. The Service Instance is similar in function to the FLEX KB discussed below, with one important distinction: whereas the FLEX KB detects database changes and updates itself accordingly, the APS Service Instance does internal processing, populates its own internal data structures, and then makes the appropriate changes to the database. There is no mechanism for detecting a change to the database from another source (the assumption being that there is no other source). Because one of the goals of the DAOC system was to provide system operability through communication outages, there is an APS database at every site, together with an APS Service Instance. (If the APS Service Instance is unavailable because of a communications failure, no planning work can be done.) To mitigate the problem caused by the Service Instance's not updating itself from the database, a Service Instance reinitialization function was introduced for DAOC. Changes made at one site are visible at the other sites regardless of whether the Service Instance has been reinitialized, because the APS screens are populated directly from the persistent storage rather than from the Service Instance. Reinitialization is required only when a remote user wishes to operate on a mission planned elsewhere (e.g., to delete, modify, or fill a support request). In addition to this, some minor changes were required to prevent collision of unique identifiers created at other sites. The method used for this was to introduce a 2-letter site identifier for each site, and block off ranges of unique identifiers for each site.

Another issue with APS integration was the use of multiple methods of reaching the database. While most of the APS database interfaces are done through dynamic SQL (using the Oracle Pro*C interface), some capabilities of APS were implemented using OraPerl (the Perl language with access to Oracle). The primary affected area is ATO Management (Creation, Deletion, etc). The OraPerl interfaces were not distributed for the DAOC prototype.

For example, since ATO Creation is implemented in OraPerl, it is not distributed. If a user creates a new ATO locally, that ATO is not guaranteed to exist on any remote nodes. Therefore, if the user wants to begin planning on the new ATO immediately, the DUS Manager GUI should be used to delay the transmission of updates to remote nodes. As the user of each remote node creates the new ATO, the DUS Manager GUI can be used to transmit the delayed updates. If the user does not delay transmitting the updates under this scenario, the databases would go out of sync (the updates would be transmitted immediately, but there would be no corresponding connection at the remote node, and the inconsistency would be reported to both sites via the DUS Manager GUI).

Performing a Target Nomination List (TNL) pull is a distributed capability, but changes to some internal tables are not distributed, owing to the use of OraPerl. This is a detail of interest to administrators and developers, but it will not be noticeable to the application user.

Applications using the DUS make a straightforward operation call, and the DUS is then responsible for handling all communication with local and remote databases.

4.3.2 FLEX Integrated into DAOC

FLEX Engineering Prototype (EP) version 2 was integrated into the DAOC system. FLEX EP2 uses embedded SQL (using Oracle's Pro*Ada interface) exclusively for database access; therefore all FLEX database access is distributed through the DUS. Because of the design of FLEX EP2, no code changes to the application were required other than replacing the Pro*Ada preprocessor with the DAOC Ada2SQL preprocessor, implemented in awk. FLEX uses database triggers to notify its knowledge base (KB) of changes to the database; the KB then updates itself directly from the database.

The data transfer tool used to populate the FLEX database from the APS database, the Air Battle Plan Transfer (ABPT), was not distributed, chiefly because of the volume of data that it transfers. The DAOC services are designed to propagate relatively small database changes to intact copies of the same database at other locations; the ABPT is essentially a database copy operation. Distributing such an operation by the "one-transaction-at-a-time" method used by the DUS is not recommended. For this reason, each site is responsible for running the ABPT tool after the ATO has been released from plans to ops.

4.3.3 ADS Integrated into DAOC

Integration of ADS into the DAOC system was performed twice: once as part of the JWID 97 DAOC capability, and later as part of the Blue Flag 97 capability. The latter instance was part of an enhancement to upgrade the integrated applications to CTAPS version 5.2; however, because this version was provided as a complete new source-code delivery — and not as deltas to the previous version — it was necessary to repeat the entire integration process. For further background information about ADS in the context of a distributed collaborative planning environment, refer to Appendix C of this FTR, "Providing a Distributed Environment for the Airspace Deconfliction System (ADS)."

The integration process was threefold: first, ADS was configured into the ClearCase configuration management system; then the application was recompiled and relinked to produce a standalone executable version on the new target workstation; lastly, appropriate application code changes were made and DAOC libraries were added into the ADS makefiles to produce a distributed ADS.

The process of configuring ADS software in ClearCase was straightforward: ClearCase provides scripts which operate recursively on a UNIX directory hierarchy to create ClearCase member elements and analogous directory structures.

Producing a standalone ADS executable was also relatively straightforward. The CTAPS ADS version 5.2, unlike the version of ADS included in the JWID 97 DAOC capability, utilizes the Imake facility. Modifying the individual ADS makefiles to reflect the various system and other directories on the target workstation was simplified, and the executable was produced after several days of effort and minor code modifications.

ADS uses embedded SQL (using the Oracle Pro*C interface) exclusively for its database access, and therefore is fully distributed by integration with the DAOC services. However, as noted in Appendix C, the assumption is made that only one operator has edit capability on a particular airspace group at any given time. This results from a series of assumptions made by the ADS system. First, checks for airspace conflicts are only done at the point of creation or change of a given airspace. Second, these checks use the in-memory copy of the data rather than the data in the persistent store. Third, there is no automated update of the display when the database changes (much like the APS Service Instance, the assumption is made that there will be only one source of changes). The ADS code was modified slightly to allow a screen refresh from the persistent store without having to reload the airspace group entirely. However, if two planners are working simultaneously in edit mode and plan airspaces which conflict with each other, when they each do a screen refresh they will see the airspaces from the other site, but the software will not detect the conflict.

Producing the distributed ADS executable entailed further modifications to the makefiles and inclusion of GOTS software libraries, most notably the Graphical Kernel System (GKS) library. Obtaining a cleanly linked executable was again straightforward; however, solving runtime problems took a greater effort. The majority of runtime problems encountered resulted from embedded SQL code that was not properly parsed by the DAOC C preprocessor. For a more detailed account of this effort, refer to Appendix E of this FTR, "SQL Conversions for ADS-DAOC Integration."

4.4 Technical Issues: JFACC Jumpstart

For details regarding the technical issues encountered during the DAOC JFACC Jumpstart effort, refer to Appendix G of this FTR, "DAOC Interim Technical Report."

5. SUMMARY AND RECOMMENDATIONS

5.1 Summary

The DAOC system went through several evolutionary stages before it reached the final prototype form that was demonstrated at Blue Flag 98-1. Since each stage was marked by a milestone exercise or demonstration, a summary of the evolution of DAOC is given here according to these milestones.

1. ***JWID 95.*** APS CAD-04 and FLEX EP1 were integrated into the system. A central database architecture was used. Application-specific CORBA services were utilized that allowed remotely running instances of APS or FLEX to access their database across the WAN. Issues arose from the central database architecture. A remote site lost all or most of its functionality during communication failures because its applications could not reach the database at the central site.

2. ***JWID 96.*** APS CAE-04 and FLEX EP2 were integrated into the system. The DAOC architecture was redesigned based on lessons learned from JWID 95. An application database was provided at every location that the application was to run, and the DUS CORBA service was introduced to keep the multiple copies of the database synchronized. The NightHawk trusted guard was integrated into the system to separate DAOC suites running on networks at different security levels.

3. ***JFACC Jumpstart.*** The JWID 96 software was integrated with the JTF ATD services in support of the JFACC Jumpstart effort. The DAOC applications took their input data from the Common Plan Representation (CPR) and output the ATO back to the CPR. The Decision Support System (DSS), a Web-based data mining and aggregation tool, was introduced.

4. ***JWID 97/DAOC ATD.*** ADS was integrated into the DAOC suite. Version 2 of the DSS was produced. Some work was done with other groups to produce a multilevel security (MLS) prototype to automate the JWID 96 activities between security levels.

5. ***Blue Flag 98-1.*** The CTAPS applications (APS and ADS) were upgraded to CTAPS 5.2 versions. FLEX was upgraded to a later version of EP2.

5.2 Recommendations

This section details issues that will need to be addressed in further DAOC work.

- Application-specific issues will have to be addressed; these usually arise from assumptions made in the individual application design which are not compatible with a distributed system. Examples of this are the Service Instance reinitialization in APS, and the failure to detect conflicts under certain circumstances in ADS.

- Since the Corbus ORB is no longer a supported product, efforts should be made to port the DAOC services to a commercial ORB that is widely used in USAF software, such as Orbix.

- Data-intensive operations which were not distributed by the DAOC prototype system should be distributed. Examples of this are the ABPT between the APS and FLEX databases, and the ATO Create and Copy operations in APS. It is not recommended that the same mechanism be used by the DAOC services for database synchronization to do this—the DUS is designed to propagate small changes to the database to remote sites, not to do massive data transfer. The recommended method would be to send a signal to the remote sites to begin the desired process; the transfer or copy process could then be started locally at all sites. Some workflow management would also be needed for this function, to prevent conflicts between automated processes and user activities.

- The DSS should be expanded, as it provides an "information anywhere" capability that requires minimal installation and maintenance (only a laptop PC, a Web browser and a network connection are required). The DSS could be expanded to provide limited data-entry capability as well.

- A collaborative tool (such as Collaborative Virtual Workspace) should be used in conjunction with the DAOC software to provide human collaboration in addition to the data-level collaboration provided by the DAOC services. Providing an additional screen for DAOC workstations to allow the operator to communicate with his counterparts at other sites without sacrificing screen space on his primary workstation should be considered.

- Work done on the trusted Oracle bridge for multilevel security should be continued and expanded. This would allow the DAOC system to operate relatively seamlessly across security boundaries.

- The DUS retry queues should be periodically written to persistent storage. This would expand the DUS's current capability to withstand communications failures, and allow it to withstand machine crashes (e.g., due to power failures) as well.

APPENDIX A
ACRONYMS

A/C	Aircraft
ABP	Air Battle Plan
ABPT	Air Battle Plan Transfer
ACC	Air Combat Command
ACDB	Air Combat Database
ACO	Airspace Control Order
ACPT	Air Campaign Planning Tool
ADS	Airspace Deconfliction System
AFB	Air Force Base
AOC	Air Operations Center
AOR	Area of Responsibility
API	Applications Programming Interface
APS	Advanced Planning System
ATD	Advanced Technology Demonstration
ATM	Air Tasking Message
ATO	Air Tasking Order
BBN	Bolt, Beranek and Newman
CAFMS	Computer Assisted Force Management System
CDB	Common Database
CIS	Combat Intelligence System
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-the-Shelf
CPR	Common Plan Representation
CRIPMH	Corbus-ized IPMH
CTAPS	Contingency Air Planning System
CVW	Collaborative Virtual Workspace
DAOC	Distributed Air Operations Center
DCE	Distributed Computing Environment
DRA	Defence Research Agency
DSS	Decision Support System
DUS	Distributed Update Server
EP	Evolutionary Prototype
FLEX	Force Level Execution System
GFI	Government-Furnished Information
GKS	Graphical Kernel System
GOTS	Government Off-the-Shelf
GUI	Graphical User Interface

ID	Identification
IDL	Interface Definition Language
IPMH	Interprocess Manager Host
JFACC	Joint Force Air Component Commander
JTF	Joint Task Force
JWID	Joint Warrior Interoperability Demonstration
KB	Knowledge Base
LAC	Local Area Controller
LAN	Local Area Network
MAAP	Master Air Attack Plan
MAP	Master Attack Plan
MCTSSA	Marine Corps Tactical System Support Activity
MLS	Multilevel Security
NRaD	Naval Air Command, Control and Ocean Surveillance Center Research, Development, Test and Evaluation Division
ORB	Object Request Broker
PDL	Program Design Language
RAAP	Rapid Application of Air Power
RDB	Reusable Database
SQL	Structured Query Language
SSS	System/Segment Specification
TAC	Tactical Air Command
TBMCS	Theater Battle Management Core Systems
TFT	Time Off Target
TNL	Target Nomination List
TOT	Time On Target
UID	Unique Identifier
USAF	United States Air Force
USMTF	United States Message Text Format
WAN	Wide Area Network

APPENDIX B
REFERENCE AND SOURCE DOCUMENTS

The following documents are the source and reference documents for the Distributed Air Operations Center Final Technical Report.

	Document Number and Date Issued	Document Title
1.	ANSI Z39.18-1987 1987	Scientific and Technical Reports — Organization, Preparation, and Production
2.	DOD-STD-2167A 29 February 1988	Defense System Software Development
3.	DOD-STD-2168 24 April 1988	Defense System Software Quality Program
4.	ESD-TR-8-001 MITRE May 1988	Software Management Metrics
5.	LITG960033 12 February 1996	Distributed Air Operations Center Software Development Plan
6.	LITG960034 9 February 1996	Distributed Air Operations Center Instruction Binder
7.	LITG960037 July 1995	Distributed Air Operations Center Program Plan
8.	LSIS900213 15 October 1990	Strategic and Information Systems Division, DOD-STD-2168A Configuration Management Program
9.	LSIS900213 1 March 1990	Strategic and Information Systems Division, DOD-STD-2168A and DOD-STD-2168, Quality Assurance Program
10.	LSIS920243 July 1994	Strategic and Information Systems Division, Software Engineering Instruction Binder (SEIB), Revision D
11.	MIL-STD-1521B 4 June 1985	Technical Review and Audits for Systems, Equipment, and Computer Programs
12.	PR NO. C-5-2236 30 June 1995	Statement of Work for Distributed Air Operations Center, Contract No. F30602-95-C-0203

APPENDIX C
PROVIDING A DISTRIBUTED ENVIRONMENT FOR THE AIRSPACE DECONFLICTION SYSTEM (ADS)

Author & Project Engineer: Francis A. DiLego, Jr.

Performing Organization: Rome Laboratory /C3AB

525 Brooks Road

Rome, NY 13441-4505

APPENDIX C
PROVIDING A DISTRIBUTED ENVIRONMENT FOR THE AIRSPACE DECONFLICTION SYSTEM (ADS)

C.1 INTRODUCTION

The goal of this project is to integrate a standalone Air Force (AF) planning application into a distributed computing environment. The level of distribution that will be attempted is first one of distributed multi-user database access over a wide area network (WAN). The functionality will be provided by the relational database management system (RDBMS) services afforded by the Cronus Distributed Computing Environment (DCE). A tighter integration between the relational/procedural AF application and the object-oriented Cronus DCE will be pursued as time permits. The intent is to enhance of the application's abilities by driving the design of the "new" application towards an object-oriented, distributed architecture.

C.2 OVERVIEW OF APPLICATION AND RELEVANT TECHNOLOGIES

C.2.1 Airspace Deconfliction System (ADS)

The Air Space Deconfliction System (ADS) provides the Air Force Modular Air Operations Center (MAOC) Combat Plans and Operations personnel in the Theater Air Control System (TACS) with an automated set of tools to enable them to rapidly and effectively deconflict airspaces. The information generated within ADS is utilized by personnel performing combat plans and combat operations functions and is ultimately transmitted as an Airspace Control Order (ACO) through the Joint Interoperability of Tactical Command and Control Systems (JINTACCS) Joint Message Preparation and Parsing (JMPP) component, or as part of an Air Tasking Order (ATO) through the Computer Assisted Force Management System (CAFMS). Failure to adequately perform the deconfliction function can result in the loss of essential combat resources [SUM 94].

The basic purpose behind ADS is to define required Airspace Control Means (ACMs) (e.g., ranges, routes, airfields) and to ensure that they do not conflict with each other. Two ACMs are in conflict if altitudes, geographic locations, and start/end date/times overlap; all three must overlap for the two ACMs to be in conflict [SDD 94].

The deconfliction process will only take place within a defined airspace group. ADS does not deconflict airspace belonging to different groups. This particular aspect of ADS is a function of concurrency control and will be viewed in more detail under the Follow-On Work section of this report.

C.2.2 ADS Software

The ADS is a monolithic X11 application. It is written primarily in C and runs on an AFCAC-308 platform (AFCAC-308 is the definition for the Air Force's tactical workstation. The current AFCAC-308 workstation is a Sun SPARCstation running Sun OS 4.1.3). There are several external separately compiled programs that ADS uses at runtime. These programs are executed via system calls at run time. These external programs are primarily used for uploading and downloading to other legacy systems within the MAOC. This up and down loading is primarily done on the data level, where the data used is static for a "snapshot" period of time. No true process level information is exchanged through these external interfaces. Internal to the application, ADS interfaces with the user through a standard X11 windowed graphical user interface. Through this interface the user can read, write and manipulate data stored in the database. ADS also has some limited mapping capability. The actual map data, which is in World Data Bank II format, is stored in the file system as a single data file. However, the lat/lon bounds of which a particular users is accessing during a planning session, are stored and maintained in the database. The final area to be addressed is the ADSs database interaction.

C.2.3 ADS Database Interaction

ADS database interaction will be a primary focus of this project. Currently, the ADS software interfaces with the database via the Oracle furnished utilities. At the code level this consists of the ProC language and pre-compiler for C programs. What a software developer would do, is write software in C and wherever database manipulations were needed the developer would embed the ProC extensions. These extensions would contain data abstraction definitions which would be converted to C data types. Other extensions are SQL statements, database control flow statements, and error handling statement. All of the above extensions are run through a ProC pre-compiler along with standard C code in which they are embedded. The output of the ProC preprocessor is standard C code, which is then compiled and linked in the standard fashion.

C.2.4 Cronus DCE

C.2.4.1 Cronus 3.0

At the highest level, the Cronus Distributed Computing Environment (DCE) provides an integration environment for the development and execution of distributed applications. Modules of the distributed application would be scattered across a diverse set of hosts. The hosts are physically interconnected into clusters by a local area network (LAN), with clusters being interconnected by a wide area network (WAN). It is assumed that standard operating systems would be executing on the different machines, and that the computers would be interfaced through standard networking protocols. The Cronus DCE would exist as a set of system software modules executing within the structure of the constituent operating systems, and in concert would provide a "virtual" application environment spanning all of the computers. The boundaries of the machines, their physical differences, their distribution could all be masked from the application developer.

The distributed computing paradigm which was chosen for Cronus was the object model. The Cronus implementation of the object model is as follows: an object is a structured block of state information, and a set of operations that provide the means to manipulate this state. Objects are thought of as active entities, like processes in traditional systems, since conceptually the objects themselves perform the operations invoked on them. Current computer architectures do not as yet directly support objects as active entities; they provide a more traditional process and memory model of computation. This leads us to the implementation of this which is the client -server approach.

Since processes are the active entities in traditional computer systems, the emulation of active objects must involve the native operating system's processes. All objects of the same type respond to the same operations, and so the process that implements objects of the same type should execute the same operation processing routines. Thus a single process, known as an object manager, implements all objects of the same type on a particular machine. The invocation of an operation on an object is implemented in Cronus as a message from a client process to the object manager for the object (server process), the processing of that operation by the manager, and the managers reply message back to the client. Cronus uses this client-server implementation technique to provide to application programs the illusion that the Cronus world consists of a collection of active objects.

In Cronus, these managers can be viewed as system level services. That is, not only would the normal system services such as directory management, process execution control, interprocess communication, synchronization, security, configuration management, etc, be supported, but because Cronus is object oriented, these services and any developed by programs would also follow as services.

In order for the client and server processes to communicate successfully, they must agree on the format and interpretation of the messages they exchange. For this reason, Cronus was designed using the well-known structuring technique of a protocol hierarchy. Each layer of the protocol hierarchy uses the lower layers of the hierarchy, along with processing and protocol information associated with the layer itself, to provide a higher-grade service to the protocol layer above. The layers of the Cronus protocol hierarchy are:

```
Application
    RPC Translation/Message Support Library (Operation Protocols)
        IPC (Peer to Peer protocol)
            Network (TCP/UDP/IP).
```

C.2.4.2 Cronus Database Services

One of the inviting features of Cronus is that it has the ability to interact with three Commercial-Off-The-Shelf (COTS) Relational Database Management Systems (RDBMS). This functionality is essential to making Cronus a comprehensive solution to building distributed applications. The reason for this is obvious in that many of today's military and commercial systems rely heavily on database management systems of one sort or another. Because of this, creating a new RDBMS capability unto itself within Cronus would be of no use. So, the approach Cronus takes to yielding RDBMS capabilities is to encapsulate the functionality of the COTS RDBMSs. Thus, Cronus can interact with the following RDBMSs: Informix, Oracle, and Sybase (ADS uses Oracle for it's RDBMS. For this reason, references to an RDBMS from here on out will pertain to Oracle).

It is important to note that because Cronus is object oriented, the only difference between interacting with one database system and another is at the database level for each individual vendor's implementation. In other words, from a Cronus perspective, one would interact with any of the three RDBMS systems in the same manner. This is achieved with a well defined object structure for databases and the operations to be performed on said object (i.e., object oriented design methodologies). This approach exploits commonalties in RDBMSs and allows for a maximum amount of software reuse through Cronus's hierarchical object typing structure, thus yielding a common interface to each of the database systems.

The Cronus approach mimics the design used by most RDBMSs in which a front end provides the client with a user interface, while the back end accesses the actual data. Each concurrently executing client program is assigned an individual database session, in whose context any queries or updates are performed. These sessions are implemented via the DBSession object type. New sessions are created as required, via the DBDefs object type. Features common to Cronus supported databases are implemented via these types. Database-specific functions are implemented by the appropriate subtypes [INTRO. 93]. In detail, the database object manager types (e.g., Oracle) are a subtype of DBDefs which is a subtype of SQLDefs which is a subtype of object. The Oracle session object is a subtype of DBSession which is also a subtype of SQLDefs and then of Object. Figure 1 depicts the Cronus object model.

The Oracle database manager is responsible for starting the lightweight processes known as *Worker* processes. Each Worker process is the front end to the database. When interfacing to the database from this point on one would communicate with the particular Worker process that managed the session object which was created for their use. As mentioned above, any subtype of SQLDefs has the same associative access capability on objects which it manages. In the case of the Oracle session object, they are the encapsulated entities of the RDBMS which gives us this bridge between the relational domain into the object domain.

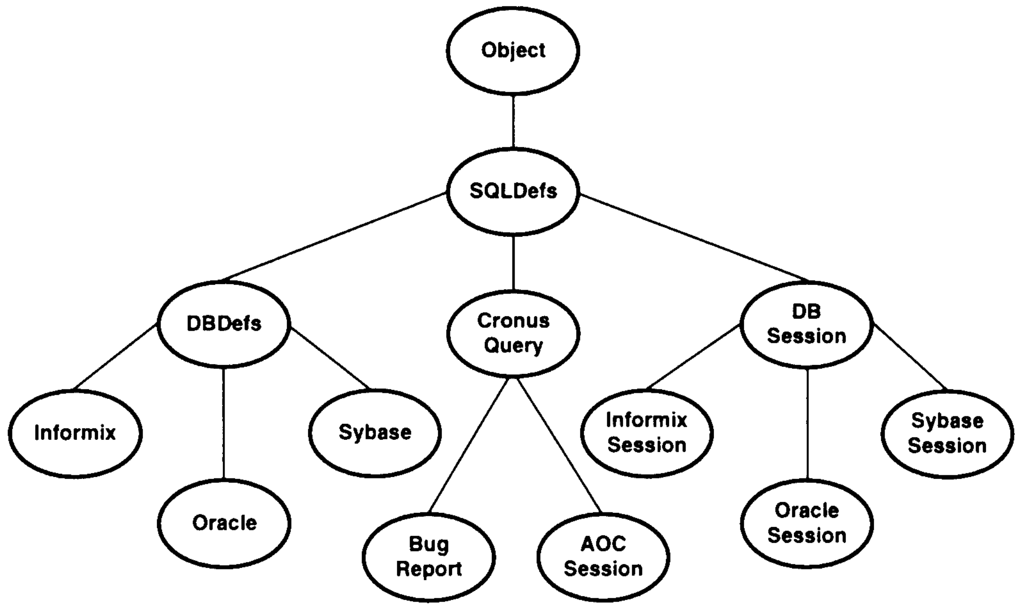

Figure 1.

Cronus also supports its own version of associative access query processing on Cronus objects. This access is through the use of an SQL-like language implementation on the Cronus type CronusQuery which is a subtype of SQLDefs. When a new object type is developed and this capability is a desired feature of that type, the developer need only make the new object type a subtype of CronusQuery. This feature greatly simplifies integrating or transferring an application from a relational domain to an object domain and vice versa.

C.3 DISTRIBUTING ADS

C.3.1 New Data Base Interaction

To implement ADS database interaction using Cronus supported services, replacement of the Oracle ProC database interface software is required. There are 30 ProC database interface source code files which will require modification. The source code in these ProC files which is straight C will be preserved to maintain existing ADS functionality. The ProC code will be supplanted by Cronus client code which interfaces with the Oracle RDBMS session object (as described earlier in Database Services section). This should be a quick and efficient first step method of bridging ADSs relational/process domain into the Cronus object domain. In this scenario, the RDBMS has not been eliminated from the application domain, ADS can now extract benefit from both domains.

There are some major differences in the two methods of programming. One aspect which is very noticeable in the ProC implementation is the use of globally defined static variables. In the case of a large database interactive transactional type of program, using this would be impractical because of the memory requirements. With the high demands for memory that today's graphical, windowed applications have on systems, an attempt to avoid heavy use of loitering static memory segment should be made.

Cronus, on the other hand, utilizes a dynamic memory implementation. Like everything else in the area of computers, this type of implementation is not without its drawbacks. The drawbacks here are that the user will have to wait at function invocation time for the memory to be allocated. However, this memory allocation is somewhat trivial, so the anticipated results at run-time will be no noticeable loss in performance. Another useful feature of Cronus is its built-in dynamic memory allocation for most functions and operations. The programmer need only supply a pointer to the type of data to be returned from an Cronus function; when the routine is finished using the data returned, the memory can be freed via another supplied Cronus function.

C.3.2 Query analysis

ADS performs many different associative access queries against, and executes many different SQL data manipulation statements on, the database. Among the type of SQL statements ADS performs are: simple multi-column select and column-based ordering; multi column select and data type conversion on received data; dynamically-generated; and, part static/part dynamically-defined.

The ADS ProC files frequently contain multiple procedures, some nested, for performing data access queries on the database. Some of the procedures require user input into the windowed application. Some of the procedures are generic queries to populate displays in the windows while others could be for number crunching routines. Whatever the procedure, the first database interaction has to be one of connecting to the database. This is where the development of a database "connection" or session manager is required.

C.3.3 AOCSession.mgr

The need for a database session manager comes from a larger scope problem. As previously mentioned, ADS exists within an AOC with other AF applications. Most of these applications have a RDBMS of one form or another. If the method of integrating ADS is applied to these other applications, something will be required to manage connections to the different databases to eliminate multiple redundant database sessions. It is obvious from this that the connection to the data base, the data base location, the database type, etc., will be transparent to the application. Therefore, for the first implementation, the primary function of the AOCSession.mgr is to maintain the AOC's applications database sessions in Cronus (for this particular instance ADS DB sessions). The ADS client software would make a request (Cronus operation invocation) to the AOCSession.mgr for a data base session UID. Once ADS receives this session UID it can then interface with the database. The client program would submit, via the operation invocation, a database login and password. The server (Cronus manager) would return to the application a valid database session UID. If the operation failed, an error message would be returned and the appropriate action could be taken. In the case of a valid session UID being returned ADS can now interact with the ADS database.

C.4 FUTURE WORK

If time permits, the access control aspect of ADS should be addressed. In order to make ADS multi-user, aspects of how ADS implements assess control must be modified. One solution is to build a Cronus object manager that will be a center of access control for certain aspects of the application. The current ADS access control schemes and ways of modifying these schemes are discussed in the following sections.

C.4.1 Current ADS Access Control

ADS currently has several forms of access/concurrency control mechanisms. These methods are too rigid and static for a system which exists in a dynamic environment like an AOC. Two of the access control mechanisms are implemented through user environment variables. The first environment variable is "OPERATING_DG". This variable is set to one of two things either " PLANS" or "OPS". This designation is appended to the table names in the data base. This would allow the user to access only those tables with this prefix designator. The second environment variable is "AUTHN_USE_CODE". This variable represents the users privileges on the tables in the database. This variable can take on the integer values 1-4. Each consecutively higher value inherits the rights from the previous value. The rights represented by the integer values are 1 = read, 2 = write, 3 = modify, and 4 = administrative. The above access control mechanisms are implemented in the user interface (UI). That is the database has no "knowledge" of there use or functionality.

Another method of access control and concurrency control which is used is that of an "in-use" column in one of the tables of the database. When a user chooses a "group" to develop air spaces within, the in-use field for that row of the table is set to "L". This signifies that it is in-use and locked out from another user to access as a group to work within. If a group has not yet been chosen for use, the in-use field would contain an "F" to signify it was free. Although this mechanism is contained in the database, it is not truly under the control of the RDBMS. The control still lies in the user interface. This is because the selection of a group through the UI initiates an SQL update operation to the table containing this field. When another user would select a group that was already chosen, that user's UI would check the in-use field, via SQL query, and do a string comparison of the value returned (either "L" or "F"). The comparison results would define the new users access to the group requested (e.g., in-use read only, not in-use up to level 3).

Other than running the whole AOC at system high, there is one more method of control. In this method, which poses high security risks, there is only one database user login and password, which are embedded in the software as an ASCII string. When a user runs ADS, ADS will login to the database as the same user/password every time. Although only one user can operate within an airspace group at one time, many different physical users can access the database at one time. From the database view, however, there is only one user accessing the database, and thus, no traceability or accountability back to an individual user is possible.

C.4.2 Proposals for New ADS Access Control Paradigms

Changing the above mentioned methods to allow multiple sessions to occur would require several things. One would be to further analyze the code to determine if shared manipulation of certain data structures would cause faulty calculations to occur. Since number crunching of spatial and temporal oriented information is one of ADSs primary jobs, this is of major concern. Another analysis would be to evaluate whether or not a reimplementation of the existing access control mechanisms would be worthwhile. A third option or analysis would be to implement an entirely new method of access control following a total object oriented implementation under Cronus. In this third case, two approaches seem feasible. The first would confine the access control mechanisms to an object design and implementation. This would keep the software a strict client application and rely on Cronus operation invocations to gain access to the database in the fine grained manner we are looking to achieve. The second method would be to fully "objectize" the application within the Cronus domain. This would require several Cronus object managers. Each manager would manage one or more object types relating to a particular feature or function that ADS performs now. Examples of such managers would include: a Map Manager; an Airspace Group Manager; an ACM Manager; and an ACO Manager. In this full object implementation, access control could be handled by the already-supplied service of Cronus via the Authentication Manager. Further granularity of access control could be accomplished at the object design phase within the type definition of the objects and their operation. With a complete object design of ADS, some of the inherent features of Cronus can be taken advantage of, one in particular being its replication of objects and their managers. Other features are parallel operation invocation through the futures mechanisms; associative access to objects by making them subtypes of Cronus Query (which could eliminate the Oracle database all together); and cluster management. Each would contribute positively to achieving the scalability, reliability, survivability, and maintainability offered by distributed computing paradigms in general.

C.5 TEST BED

The testbed consists of a cluster of machines running within a Cronus configuration. This configuration consists of multiple heterogeneous hardware and operating system platforms, including SparcStation 20 workstations, Solbourne multi processors, SparcStation 360 servers. The configuration spans two local area networks connected by a wide area network. One 360 server is host to the Oracle RDBMS which contains the ADS database. Because of the configurable nature of Cronus, the AOCSession.mgr and "N" ADS clients can run on any Cronus host in the configuration. The Oracle database manager does need to run on the machine hosting Oracle. The configuration set-up is depicted in Figure 2.

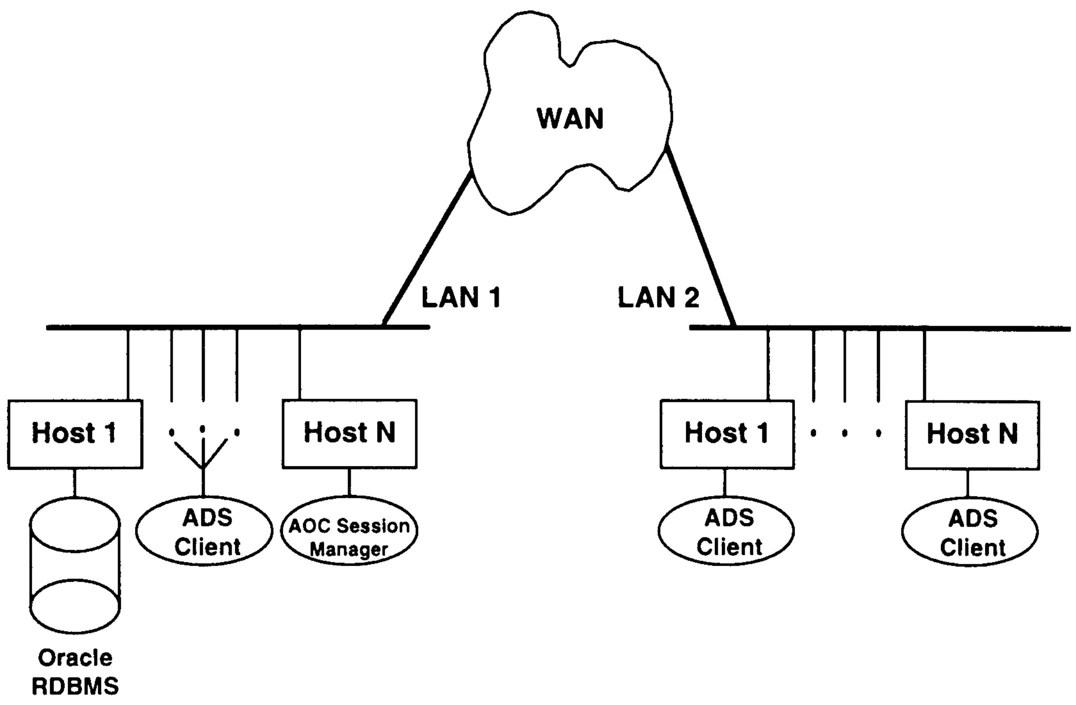

Figure 2.

C.6 OBSERVATIONS/EXPERIENCES

This section details lessons learned from the initial exercise of integration ADS into a distributed environment.

C.6.1 Test Bed

Setting up a testbed in which to work is a large task in and of itself. There are many things to consider when trying to achieve this seemingly trivial task. Things like system types, operating systems, disk space, memory, graphic packages, software support packages, compilers, TIME, etc.

C.6.2 Software Engineering Standards

When using operational military software, standard programming practices are not often closely adhered to. This results from the real-time need users of the system have for operational software and an often unrealistic time to provide it. As a result, developers will do what ever it takes to make the software work properly and deliver it in a timely manner, even if it means straying from normal software engineering practices. A good example of this is makefiles. Manual modifications not consistently applied resulted in explicit paths and software that need libraries linked in a certain order for one section of code and another order for a different section of code.

C.6.3 Programming Styles

In this software, I came across several different programmers and their styles of coding, none of which were alike. This makes ones task even more complicated when trying to make modifications to an 30k lines of code application. A good example encountered of this is the varied and inconsistent modification and use of header files. Another good example was the level of commenting and documentation.

C.6.4 Estimation of Effort

There are always undocumented features of a system that will contribute to the time require to complete the work. ADS was not an exception. Remember to incorporate this in estimates of effort and cost.

C.7 REFERENCES

[SUM 94] Air Combat Command Computer Systems Squadron, "Software Users Manual for the Airspace Deconfliction System of the Contingency Theater Automated Planning System", Langley AFB, VA, 1 July 1994.

[SDD 94] 1912 Computer Systems Group, "Software Design Document for the Airspace Deconfliction System of the Contingency Theater Automated Planning System", Langley AFB, VA, 29 April 1994.

[Intro 93] James C. Berets, Natasha Cherniack, Richard M. Sands, "Introduction to Cronus", BBN Systems and Technologies, Cambridge MA, January 1993.

APPENDIX D
THE ROLE OF DISTRIBUTED PLANNING IN TOMORROW'S C4I ENVIRONMENT

The document for this appendix is marked "Distribution authorized to DOD & US DOD contractors only". Use the following "Report Documentation Page" sec 12a information to acquire a copy.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE February 1997	3. REPORT TYPE AND DATES COVERED In House Jan 95 - Oct 95

4. TITLE AND SUBTITLE

DISTRIBUTED COMPUTING AND C4I

5. FUNDING NUMBERS

PE - 62702F
PR - 5581
TA - 28
WU - 24

6. AUTHOR(S)

Francis A. Dilego, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rome Laboratory/C3AB
525 Brooks Road
Rome, NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/C3AB
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-96-130

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Francis A. Dilego, Jr.,(315) 330-3681

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution authorized to DOD & US DOD contractors only; Critical Technology; Feb 97. Other requests shall be referred to RL/C3AB, Rome, NY.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

ABSTRACT LIMITED

14. SUBJECT TERMS

DISTRIBUTED COMPUTING, DISTRIBUTED SYSTEMS, CRONUS, CTAPS, DISTRIBUTED AIR OPERATIONS CENTER

15. NUMBER OF PAGES

44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

APPENDIX E
SQL CONVERSIONS FOR ADS-DAOC INTEGRATION

Basic SQL to CDB API conversions

The following document is provided to explain the required CDB_API calls for the shown SQL statements.

1) INSERT/UPDATE/DELETE

The CDB_API calling structure for the SQL statements INSERT, UPDATE, and DELETE all follow the same general template. First, the argument list is constructed for each host variable within the SQL statement. Each host variable requires a minimum of four calls to populate the argument list. The first call sets the indicator type to not used. Use of indicators is not discussed in this document. The second call sets the size in bytes of the host variable. The third call identifies the type of the host variable. And the fourth call passes the host variable pointer.

NOTE: the cdbVarChar host variable type should be used in place of the cdbChar type, since cdbVarChar will work with both VARCHAR and CHAR table definitions. All string host variables MUST be NULL terminated. If you must use a single byte variable (char) use either an int definition or a VARCHAR definition with a size of 2 as shown below.

```
EXEC SQL BEGIN DECLARE SECTION;
static char         byte_value;   DO NOT USE THIS FORM
static VARCHAR                    byte_value[2]; USE THIS FORM
static int          byte_value;   OR THIS FORM
EXEC SQL END DECLARE SECTION;
```

SQL STATEMENT FORMAT:

The following sample SQL INSERT statement is presented as a typical example. The associated SQL host variable definitions are provided for clarity.

```
EXEC SQL BEGIN DECLARE SECTION;
static VARCHAR     String_1[20];
static int    Int_2;

...

static int    Int_n;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT TABLE_XXX VALUES (:String_1, :Int_2, ..., Int_n);
```

CDB_API CALL LIST:

The following function calls correspond to the above listed SQL statement.

Note that for each host variable four calls are made. Additionally, the cdbSetArg function returns a value of either SQLOK or SQLARGERROR of type cdbSQL_ERROR. All host variables that appear within the SQL statement must be placed into the argument list prior to executing the SQL statement, in this case the cdbInsert function call. The insert function call consists of the SQL statement in which the host variable names have been substituted with their C language formatting identifiers, followed by the number of host variables in the argument list. The jump label ERROR should be a series of C language statements at the end of the routine for error handling.

```
cdbSQL_ERROR cdbSQL_ERROR_Status;

if(cdbSetArg (ptrType, CDB_IND_ARG_IS_OK) != SQLOK) goto ERROR;
if(cdbSetArg (intType, sizeof(String_1)) != SQLOK) goto ERROR;
if(cdbSetArg (intType, cdbVarCharType) != SQLOK) goto ERROR;
if(cdbSetArg (ptrType, String_1) != SQLOK) goto ERROR;

if(cdbSetArg (ptrType, CDB_IND_ARG_IS_OK) != SQLOK) goto ERROR;
if(cdbSetArg (intType, sizeof(Int_2)) != SQLOK) goto ERROR;
if(cdbSetArg (intType, intType) != SQLOK) goto ERROR;
if(cdbSetArg (ptrType, &Int_2) != SQLOK) goto ERROR;

...

if(cdbSetArg (ptrType, CDB_IND_ARG_IS_OK) != SQLOK) goto ERROR;
if(cdbSetArg (intType, sizeof(Int_n)) != SQLOK) goto ERROR;
if(cdbSetArg (intType, intType) != SQLOK) goto ERROR;
if(cdbSetArg (ptrType, &Int_n) != SQLOK) goto ERROR;

if((cdbSQL_ERROR_Status = cdbInsert (NULL, FALSE,
    "INSERT TABLE_XXX VALUES ('%s', '%d', ..., '%d'", n)) == SQLOK
    || cdbSQL_ERROR_Status == SQLNOROWSAFFECTED)
    continue;
else if (cdbSQL_ERROR_Status == SQLATENDOFDATA)
    break;
else if ((cdbSQL_ERROR_Status == SQLWARNING) ||
        (cdbSQL_ERROR_Status == SQLUNIMPLEMENTED) ||
        (cdbSQL_ERROR_Status == SQLDUPLICATECONNECTION))
    goto CONNECTION_ERROR;
else
    goto SQL_ERROR;
```

The UPDATE and DELETE SQL statements follow the same general calling sequence in the CDB API and will therefore not be demonstrated here.

The next SQL statement format addressed is the SELECT. Particularly its use in a cursor. All SELECT statements should be viewed as CURSOR statements for simplicity. The following

CURSOR example shows it use in relationship to the SELECT statement. The declared cursor will return COLUMN_1 – COLUMN_n from the table where the COLUMN_NAME is equal to the supplied host variable name. After the cursor has been declared, it is opened. A series of data fetches is made until no more data is returned. The cursor is then closed.

NOTE: the cursor template should be used as presented even if only a single row return is expected. Modifications to the database later in development may cause multiple rows to be returned.

1) CURSORS:

SQL STATEMENT FORMAT:

The following CURSOR declaration is typical of those used in most data selection processes. Although simple in construction, it contains all the possible components for a statement of this type.

```
EXEC SQL DECLARE CURSOR_NAME CURSOR FOR
    SELECT COLUMN_1, COLUMN_2, COLUMN_3, ... COLUMN_n
    FROM TABLE_NAME
    WHERE COLUMN_NAME = :Data;

EXEC SQL OPEN CURSOR_NAME;

LOOP
    EXEC SQL FETCH CURSOR_NAME INTO :String_1, :String_2, :Int_3,
        ..., :Int_n;
END LOOP

EXEC SQL CLOSE CURSOR_NAME;
```

CDB_API CALL LIST:

The following source code represents the conversion of the above listed SQL statements into their equivalent CDB API calls. In the first step the cursor is declared using the cdbDeclareCursor command. It should be noted that where the host variable name appeared in the SQL statement a format identifier, '%s' now appears. The format type, ie %s, %d, %f, etc., is that same as the type of the host variable it replaced. Prior to opening the cursor, the value of the host variable must be pushed onto the argument list. This is accomplished using the same four call sequence examined in the above INSERT/UPDATE/DELETE example. Next, the cursor is opened by making a call to the cdbOpenCursor function.

Now, the basic infinite loop structure is established. Within the loop the argument list for the data to be returned is constructed using a four call template similar to the one used before. The one difference is that the indicator name has changed from CDB_IND_ARG_IS_OK to UNIVERSAL_IMPLICIT_IND. As with the other argument lists a four call sequence is required for each data value (column or field) returned. Finally, the data fetch is made by calling cdbFetchCursor. This call includes the number of columns or fields that will be returned and it must be the same number as the arguments placed on the stack. The loop structure will continue to return data, on row at a time until no more data is available. When this occurs the looping will be stopped by a break command and a call to close the cursor is issued using cdbCloseCursor.

```
if(cdbDeclareCursor (NULL, "SELECT COLUMN_1, COLUMN_2, COLUMN_3, ...
    COLUMN_n FROM TABLE_NAME WHERE COLUMN_NAME = '%s'",
    &CURSOR_NAME) != SQLOK) goto ERROR;

if(cdbSetArg (cdbIndicatorType, CDB_IND_ARG_IS_OK) != SQLOK) goto ERROR;
if(cdbSetArg (intType, sizeof(Data)) != SQLOK) goto ERROR;
if(cdbSetArg (intType, cdbVarCharType) != SQLOK) goto ERROR;
if(cdbSetArg (ptrType, Data) != SQLOK) goto ERROR;

if(cdbOpenCursor (CURSOR_NAME, 1) != SQLOK) goto ERROR;

for(;;)
{
    if(cdbSetArg (ptrType, UNIVERSAL_IMPLICIT_IND) != SQLOK)
        goto ERROR;
    if(cdbSetArg (intType, strlen(String_1)) != SQLOK) goto ERROR;
    if(cdbSetArg (intType, cdbVarCharType) != SQLOK) goto ERROR;
    if(cdbSetArg (ptrType, String_1) != SQLOK) goto ERROR;

    if(cdbSetArg (ptrType, UNIVERSAL_IMPLICIT_IND) != SQLOK)
        goto ERROR;
    if(cdbSetArg (intType, sizeof(String_2)) != SQLOK) goto ERROR;
    if(cdbSetArg (intType, cdbVarCharType) != SQLOK) goto ERROR;
    if(cdbSetArg (ptrType, String_2) != SQLOK) goto ERROR;

    if(cdbSetArg (ptrType, UNIVERSAL_IMPLICIT_IND) != SQLOK)
        goto ERROR;
    if(cdbSetArg (intType, sizeof(Int_3)) != SQLOK) goto ERROR;
    if(cdbSetArg (intType, intType) != SQLOK) goto ERROR;
    if(cdbSetArg (ptrType, &Int_3) != SQLOK) goto ERROR;
    ...

    if(cdbSetArg (ptrType, UNIVERSAL_IMPLICIT_IND) != SQLOK)
        goto ERROR;
    if(cdbSetArg (intType, sizeof(Int_n)) != SQLOK) goto ERROR;
    if(cdbSetArg (intType, intType) != SQLOK) goto ERROR;
    if(cdbSetArg (ptrType, &Int_n) != SQLOK) goto ERROR;

    cdbFetchCursor (CURSOR_NAME, n);
```

```
        if((cdbSQL_ERROR_Status = cdbFetchCursor (CURSOR_NAME, n)) ==
            SQLOK || cdbSQL_ERROR_Status == SQLNOROWSAFFECTED)
            continue;
        else if (cdbSQL_ERROR_Status == SQLATENDOFDATA)
            break;
        else if ((cdbSQL_ERROR_Status == SQLWARNING) ||
            (cdbSQL_ERROR_Status == SQLUNIMPLEMENTED) ||
                (cdbSQL_ERROR_Status == SQLDUPLICATECONNECTION))
            goto CONNECTION_ERROR;
        else
            goto SQL_ERROR;
    }

    if(cdbCloseCursor (CURSOR_NAME) != SQLOK) goto ERROR;
```

The two samples listed above are designed to provide the user with a basic knowledge of the CDB API calling structure as it pertains to SQL statements. This document does not cover API calls that replace the PREPARE SQL statement. Although the API supports that calling structure the awk scripts which perform the translation from SQL to the API are currently unreliable. Therefore it is believed that users should adhere to the basic templates set forth in this document when using the awk scripts to convert SQL to the API.

APPENDIX F
JFACC JUMPSTART HOTWASH EVALUATION

APPENDIX F
JFACC JUMPSTART HOTWASH EVALUATION

This document gives a detailed evaluation of the JFACC Jumpstart system from the point of view of the DAOC application. It is divided into three parts: 1) evaluation of the JFACC Jumpstart version of DAOC, 2) evaluation of the Jumpstart system, and 3) general and technical recommendations.

F.1 THE JFACC JUMPSTART IMPLEMENTATION OF DAOC

The role of DAOC in JFACC Jumpstart was a special case compared to most of the other applications. First, DAOC was intended to play a supporting role to the other JJS applications, providing execution/engagement level planning which would be used by the other applications to show the progress of the plan. Second, DAOC was not a new application created for JJS, but rather an existing suite of programs which integrated legacy planning and execution monitoring systems into a distributed environment. Third, while DAOC did interface with the Common Plan Representation (CPR) through the Plan Server, the majority of DAOC's processing was accomplished against an RDBMS which contained the data read from the CPR. Finally, DAOC operates at only one level of the planning process—the execution/engagement level. The third point in this list is the only one which was driven by technical factors; it is addressed in Section 2 below.

JJS DAOC consisted of the following components: 1) the original DAOC software, consisting of a set of CORBA services to provide data sharing among geographically separate sites, and the legacy systems APS (mission planning) and FLEX (execution monitoring) which have been integrated with the DAOC services; 2) a set of utilities for interfacing DAOC with the CPR: The Target Alerter, which waits for notification from JOE OP (through an event list) that a target set is ready for mission planning and pulls the target set into the APS database when notification is received; Mission Publication, which pulls mission data from the APS database and creates mission objects in the CPR; Mission Status Update, which pulls mission data which has been updated through the execution monitoring tool from the FLEX database and updates existing mission objects in the CPR; 3) The Decision Support System (DSS), an Intranet Data Mining Tool which queries the FLEX and APS databases at user request and graphically displays information about the status of the executing plan.

F.1.1 The DAOC Planning System

The need to use legacy applications to perform the mission planning tasks introduced some limitations to the system within the context of strategy-to-task planning. For instance, although objective data and links from target sets to objectives had been imported into the APS database, APS itself is not capable of using those relationships. Thus during the planning process this information is invisible to the user. For the Jumpstart demo the objective relationships were coded into the APS database using unrelated fields to work around this limitation. Having the objective relationships visible in some way during the planning process was an important part of building missions for the JJS demo. FLEX was likewise unable to display objective relationships for missions being monitored. The DSS, which is not a legacy application, was able to display execution status information by objective and to show the objective(s) which a mission supported when information about individual missions was requested. The way in which it did this will be addressed later in this paper.

In order to approximate a continuous planning environment, some changes or enhancements were made to APS and FLEX. The limitation in APS of being tied to a 24-hour planning cycle was removed; the planning cycle could be of arbitrary length-for the demo it was 5 days. Target sets could be imported into the planning environment and have missions scheduled against them at any time, even after the execution of the plan had begun. However, APS has no concept of current time, so the planner would not be prevented from scheduling prosecution of a new target for a time already past. FLEX likewise could handle a planning cycle of arbitrary length and could import incremental changes from APS at any time.

Data which is used by DAOC is stored in three separate places: the CPR (accessed through the Plan Server), the APS database and the FLEX database. Data transfer "bridges" from one source to another were built, however there is no mechanism to automatically keep the three in sync. (The exception to this is the Target Alerter tool, which will automatically transfer data from the CPR to the APS database upon receipt of an event.) This means that there are various "snapshots" of the data in different places which will not change until a human intervenes to run a data transfer tool. The biggest disconnect was with publication of mission status updates. In an executing plan, mission status updating is a continuous process, and there is no clear point at which to publish the updates to the CPR. In a truly continuous planning environment, there would likewise be no triggering event to begin transfer from the APS (planning) to FLEX (execution monitoring) databases.

The rather loose way in which DAOC is integrated with the CPR results from a missing feature in the Plan Server which is documented in the next section.

A related but broader issue which was not addressed during the demonstration is to decide when the Dynamic Execution Order (DXO, which replaces the old-style ATO) should be released to the units. The nature of continuous planning removes the natural "freeze point" that the 24-hour cycle provided; a decision has to be made whether to release the DXO at pre-determined intervals, at times triggered by other measures such as number of missions planned, or simply to trickle in changes as they occur. Each of these options has its own problems, but this question is a vital one if continuous planning is to be more than a demonstration concept.

In general, DAOC accomplished what it set out to do for JFACC Jumpstart. However, its capabilities were under-utilized for the demo. No true continuous planning was done, the distributed nature of DAOC was used in a limited fashion but not exhibited, and DAOC's outputs of mission objects to the CPR were not used by OSA or CA, the intended consumers.

F.1.2 The Decision Support System (DSS)

The DSS provides a variety of views into the status of the executing plan which are presented to the user in graphical format through any standard Web browser (Netscape was used for the

Jumpstart demo.) It was developed for JJS with a view towards continuing its use for future DAOC activities (for instance, it will be shown at JWID 97.) Because DAOC is concerned only with the execution/engagement level of planning, the functionality of the DSS is confined in most cases to that level of data. (The exception to this is the Objectives Explorer, which allows access to mission data from any level of objective.) The goal for the DSS is to provide, by aggregation of data, a quick way of discovering patterns of events as the plan executes. For instance, a low success rate for HARM carriers against a certain type of SAM site, a high incidence of delayed and canceled missions from a particular unit, or a particular objective which is not being successfully executed. The framework of the system is sound and allows the user a high degree of control over the data displayed without requiring a familiarity with the underlying data structures. There is much functionality still to be added, including expanding the available set of queries, adding more drill-down capability, and expanding the capability to move from one view of a set of data to another.

The integration of objective-to-task structure into the DSS was implemented by pulling the objective data from the CPR into the FLEX database with a data transfer tool. The relationships from targets and missions to objectives were already in the database from the planning activities. This mechanism allowed full query capability on the portion of the CPR at the execution/engagement level. (More on this in the next section.) Of course, the same problems noted above with respect to data transfer between two different data sources apply: If objectives changed or were created, the DSS would not reflect this until the data transfer tool was run again. This did not become an issue during the Jumpstart demo since the objectives were fixed at the outset.

F.2 THE JFACC JUMPSTART SYSTEM

This section details limitations of the JFACC Jumpstart system from DAOC's point of view. It also includes some general observations which did not affect DAOC directly.

F.2.1 The Plan Server

The Plan Server was the only one of the JTF ATD services with which DAOC interfaced directly, and so it is the only service evaluated here.

The Plan Server is a database with no query capability. The sort of data aggregation which the DSS performs would be extremely cumbersome if not impossible to perform directly on the Plan Server, since it is necessary to walk through the web of objects repeatedly to perform even a simple example. It is for this reason that the DSS requires a data transfer from the CPR into an RDBMS in order to show objective-based aggregated data. There are other weaknesses of the Plan Server as compared to commercial object-oriented database systems: There is no support for transaction management, no version control and no indexing mechanism.

The management of objects through the Plan and Web Servers is still primitive. There is no good delete capability, changes to objects create copies of objects, and one corrupt object makes the Plan Server unusable until it is removed from the Plan Server database. During the Jumpstart demo, there were approximately 4,500 objects linked to the Plan, but there were at least 16,000 objects in the Plan Server database. (The number was between 16,000 and 18,000.)

During the course of the Jumpstart final IFD and demo, it was observed that operations on the Plan Server sometimes returned very quickly and sometimes took minutes to return. From this and other anomalous behavior the developer group was able to diagnose that the Plan Server was crashing frequently and re-initializing when the next request against it was made (thus explaining the longer response times.) It is unknown why the frequent crashes were occurring; a reasonable theory would be that it had to do with so many applications accessing it simultaneously and possibly the size of its database. This behavior was not observed in DAOC's development environment.

Finally, the Plan Server was supposed to have a query node capability which was never made available. Despite the similar name, this is different from the lack of query capability mentioned above. A query node is an object in the Plan Server which, instead of containing data which is stored in the Plan Server database, contains instructions on how to build the object from other data sources. When such an object is requested, the Plan Server would go through the Data Server to retrieve the object data from the other data sources (assumed to be RDBMSs), thus providing a tight integration of the CPR with the other data sources. The lack of this capability is notable because it would provide a clear path of migration for legacy systems from their existing data sources to the Plan Server. DAOC had intended to use query nodes for this purpose; the data transfer mechanisms that were finally implemented were workarounds made necessary by the unavailability of this feature.

F.2.2 Data Sharing Problems

Despite the notion of the Common Plan Representation, there were major disconnects in data sharing during the Jumpstart demo. These problems can be roughly divided into three categories: 1) lack of inter-application communication, 2) multiple data sources and 3) design of the C2 Schema.

Lack of communication among applications took several forms. One was under-utilization of services such as event lists and triggers. Some applications were using event lists to receive messages, however they were not using triggers on the lists to notify them of new events. Instead, they were polling the event list at regular intervals, thus producing response-time delays and increased overhead for their applications. Another was produced by the caching mechanism used by some applications for CPR objects, which prevented them from seeing changes made by others to those objects without manually refreshing the cache. Finally, there was data available which was simply not used; in DAOC's case, no other application made use of the mission data in the CPR for assessment or any other purpose.

The issue of multiple data sources in relation to DAOC has already been addressed above. In addition to those comments, there were other applications making use of alternative data sources which were out of sync with each other and with the CPR. The idea of a common plan was not fully realized because of this.

The C2 Schema has the characteristic of being designed in a piecemeal fashion, with each application essentially operating off of a different subset of objects. The majority of object types were not used in the Jumpstart demo-while there are approximately 150 object types defined, only about 30 of them had instantiations linked to the plan. In addition, there are concepts such as objective or target priority which are not adequately represented in the schema.

F.3 RECOMMENDATIONS

The following recommendations are made in the spirit of constructive criticism.

Requirements for each IFD and the final demo were never pinned down. The final demo was successful because the integration team looked at what there was and built a demo out of it, not because what there was met previously defined requirements. There were several consequences of this, including the lack of inter-application communication detailed above and the fact that IFDs became development fests rather than integration exercises. Lack of interface requirements made building the interfaces a moving target, as each application was free to change things without notice for their own convenience. Two things will be needed to prevent this in future JFACC IFD/demos: 1) a set of defined functionality and interface requirements must be developed for each application for each IFD or demo, and 2) evaluations of each application's conformity to these requirements must be made before the start of IFDs.

Test data was a constant problem. Code tested against dummy test data made up by developers is notorious for failing when faced with real data. The fact that the C2 Schema was changing constantly made the data problem worse, as Plan Server data which had been produced with an old version of the schema was worthless with the new version. A good set of test data (preferably that which will be used in the demo) should be generated early in the development process and made available to all development teams so that data surprises right before the demo are minimized.

The JTF ATD services are not ready for prime time. The Plan Server has many problems with design and functionality that make it inadequate to the task it has been assigned to. Careful consideration should be given to replacing the Plan Server with a commercial object-oriented database. As noted above, with the current system there is no smooth migration path from legacy data sources to the CPR. Query nodes in the Plan Server were supposed to fill that gap. Note, however, that query nodes as specified for the Plan Server had limited functionality since they were to work only in one direction (changes to the Schema object would not be rippled to the original data sources.) If a commercial product is chosen, it may be worth looking at federated database systems such as UniSQL, which combine an object-oriented database with the ability to build objects from data stored in RDBMSs or even flatfiles. Oracle 8, when released, should have similar capabilities.

The question brought up above of when is the DXO sent to the units must be resolved.

The C2 Schema should not be accepted as it stands for the continuing work on the JFACC program. The lessons learned from Jumpstart should be used to revisit the schema and hammer it into a more cohesive shape for the next phase of JFACC. Also, the earlier that the schema is in a state where only minor corrective changes will be made the better; the constant changes to the schema which occurred during the Jumpstart development process caused many time-consuming adjustments on the part of the developers.

TECHNICAL INFORMATION REPORT FOR THE DISTRIBUTED AIR OPERATIONS CENTER

CONTRACT NO. F30602-95-C-0203

CDRL SEQUENCE NO. A005

Prepared for:

Rome Laboratory
525 Brooks Road, Bldg 3
Rome, NY 13441-4505

Prepared by:

Logicon Incorporated
222 W. Sixth Street
San Pedro, CA 90731

Table of Contents

Distributed Air Operations Center (DAOC)

Joint Forces Air Component Command (JFACC) Jumpstart Implementation

Introduction

This document provides a description of the Distributed Air Operations Center (DAOC) Integration with the Joint Task Force (JTF) Advanced Technology Demonstration (ATD) services (also known as the Global Command and Control System Leading Edge Services (GCCS LES) services) for the Joint Forces Air Component Command (JFACC) Jumpstart Program. Section 1 provides a background description of the DAOC as implemented for Joint Warrior Interoperability Demonstration (JWID) '96. Section 2 describes the changes made to the existing DAOC for integration into JFACC Jumpstart.

1. DAOC Background

DAOC provides distributed collaborative planning capability for existing Contingency Theater Automated Planning System (CTAPS) and Theater Battle Management Core Systems (TBMCS) applications. It uses a suite of Common Object Request Broker Architecture (CORBA) services designed to provide asynchronous data replication among geographically separate sites. The applications which are integrated with the DAOC services maintain their usual look, feel and functionality—data replication is transparent to the user. In addition, the DAOC services are robust through communications failures. Unstable or non-existent network connections to other sites do not affect the operability of the applications, and the databases are re-synchronized when communications are re-established.

The DAOC services consist of the Distributed Update Server (DUS), the Reusable Database (RDB) Session Server, and the Common Database Applications Program Interface (CDBAPI). All of these services are implemented using BBN's Corbus Object Request Broker (ORB). In addition to these services the CORBUS database managers (e.g., the Oracle manager and Sybase manager) are used. A brief description of the services follows.

The DAOC RDB Session Server acts as an interface between the application code and the applications database, via the CORBUS database managers. There is one RDB Session Server per actual database. Application database operations are invoked on the RDB Session Server, which then calls the appropriate database manager and applies the operation to the application database. The Session Server provides a uniform interface regardless of database type. In addition, the Session Server does not discard database connections when the user logs off, allowing the session to be recycled if another request for the same username/password pair is made. This can improve performance for applications which do multiple connect/disconnects to the same database username/password. A feature of the Session Server which is not used by DAOC is the ability to allow multiple users to share the same database session. This is deemed too dangerous for DAOC's purposes, as it allows interleaved transactions and can play havoc with rollback or commit operations. This feature is designed for read-only access.

The Common Database Applications Program Interface (CDB API) is a client library which allows application interface with the Session Server. It provides API functions which are designed to mimic the functions of ANSI SQL, e.g., declaring a cursor, opening the cursor, fetching from the cursor, closing the cursor, and transaction rollbacks and commits. A pre-processor is provided to translate embedded SQL into CDB API calls.

The DUS provides asynchronous data replication among application databases. It is designed to work with the RDB Session Server. The Session Server checks for an active DUS upon initialization, and will use the DUS if it is found. All write actions on the application database are captured by the Session Server and sent to the DUS, which then propagates these changes to other instances of the same application database. A diagram of the application interaction with the Session Server and the DUS is provided in Figure 1. The DUS provides robustness through communications failures by re-trying message sends until they are successful.

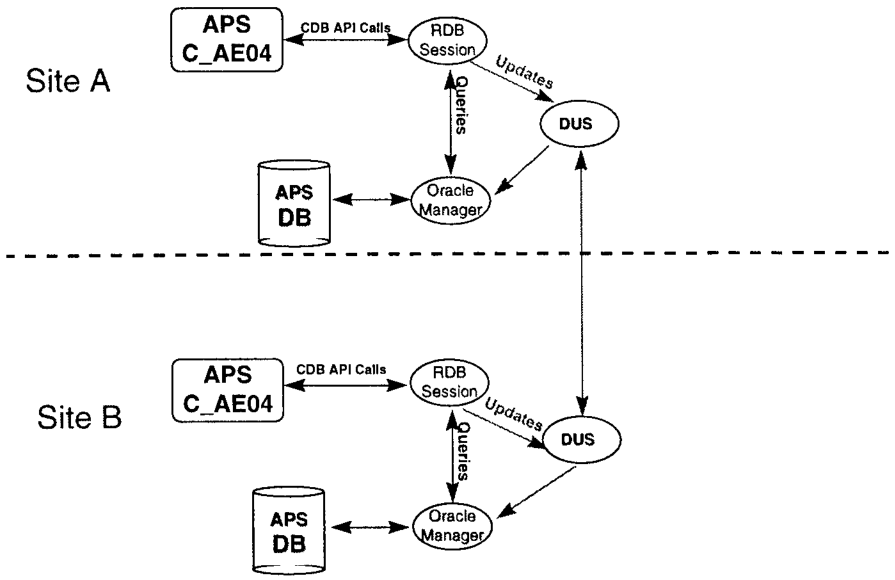

Figure 1. DAOC Database Access Interfaces

When integrated with an application, the suite of DAOC services provides a location-independent shared data space available to all users of the application at any site. Data changes initiated at other sites are available to the user in near-real-time by whatever mechanism the application normally shows data updates—in some cases, automatically and in others at user request.

2. The JFACC Jumpstart Implementation of DAOC

DAOC was tasked to provide mission planning and execution monitoring functionality for the JFACC Jumpstart (JJS) demonstration. JJS DAOC consisted of the following components:

1) The original DAOC software, described above, integrated with the Advanced Planning System (APS) (for mission planning) and the Force Level Execution System (FLEX) (for execution monitoring);

2) A set of utilities providing an interface between DAOC and the JFACC Common Plan Representation (CPR);

3) The Decision Support System (DSS), an Intranet Data Mining Tool which queries the FLEX database at user request and graphically displays aggregated information about the status of the executing plan.

The planning functionality of JJS DAOC used the existing capabilities of the DAOC services and the integrated versions of FLEX and APS. Functional differences between plain DAOC and JJS DAOC fall into 2 categories:

1) Modifications to the applications to support a non-standard Air Tasking Order (ATO) "day," i.e., one longer than 24 hours, and

2) Modifications to provide the data interface with the JFACC CPR.

JFACC Jumpstart as a whole deals with four levels of the air campaign planning process: Campaign Development, Force Application, Mission Integration, and Execution/Engagement. By the nature of the services which DAOC provided, it is only involved on the Execution/Engagement level of planning, and thus is invoked fairly late in the planning process. The Joint Objectives Editor/Operations Planner (JOE OP) notifies DAOC when an objective has been approved for planning. At this point, the target set associated with the approved option for that objective is pulled into the APS database from the JFACC CPR. (The target sets are put into the CPR by Objectives/System Analysis (OSA).) Once this is done, planning in APS can occur as usual. New target sets can be imported at any time, and the planning period of APS is no longer constrained to 24 hours. When missions are planned against the imported targets, they are written back to the CPR and are linked back to the objectives which they support. (Attack missions support the objectives which the target(s) they are prosecuting support; support missions support the objectives which their receivers support.) The link between the planning tool (APS) and the execution monitoring tool (FLEX) has been modified so that planned missions can be pulled into FLEX for monitoring as they are planned, rather than only at the completion of the planning process. Execution status information entered through FLEX is written back to the JFACC CPR in the form of updates to the existing mission objects. In addition, notices of mission creation or changes are placed in an event list for monitoring by other applications. The Jumpstart mission planning process is illustrated in Figure 2.

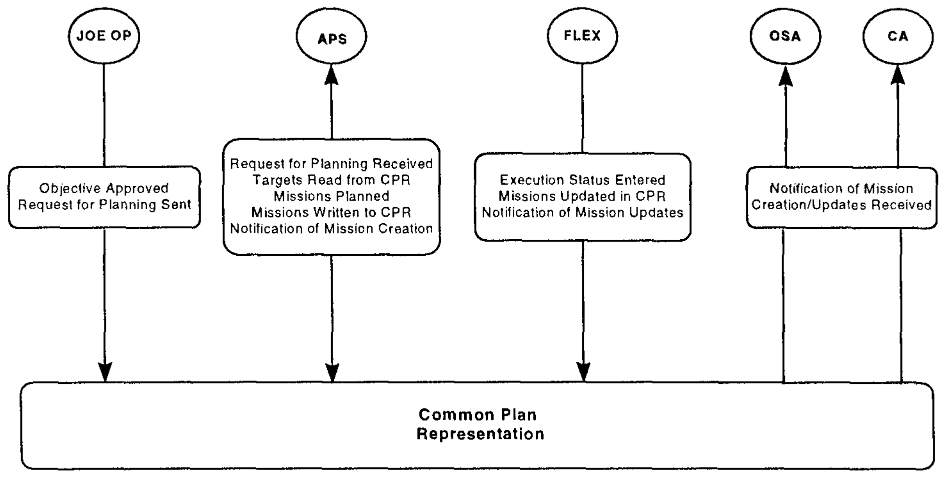

Figure 2. DAOC JJS Planning Functional Flow

The technical implementation of JJS DAOC consisted of a set of utilities to effect the integration with the JFACC CPR. Three utilities were provided:

1) The Target Alerter,

2) Mission Publication, and

3) Mission Update.

The Target Alerter is a client to both Orbix, used by the JTF ATD services, and CORBUS, used by the DAOC services. It runs continuously and registers interest in an event list (TargetListReady) which is used by JOE OP to signal DAOC that a target set has been approved for planning. When the Target Alerter receives an event in this list, it first interfaces with the JTF ATD Plan Server to retrieve the approved target set, and then interfaces with the RDB Session Server and the DUS to propagate the targets into the APS databases at each DAOC site. This replaces the normal APS Target Nomination List (TNL) pull from the Rapid Application of Air Power (RAAP) database. DAOC had previously modified this data transfer to pull from the RAAP database into temporary tables in the APS database, and then use the normal APS functionality to pull the targets into the ATO-specific areas of the database. Additions to this mechanism were made for JFACC Jumpstart to keep track of the Orbix object references for the targets and objectives which they support in the database. This data is used by the Mission Publication and Update utilities when writing mission objects back to the CPR. All these functions occur through the DAOC Services, and thus are distributed to all instances of the APS database. The Target Alerter interfaces are illustrated in Figure 3 below.

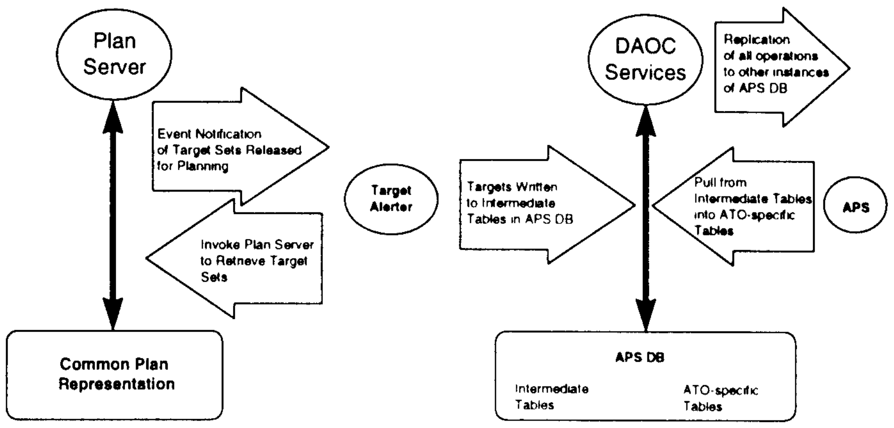
Plan Server
Event Notification of Target Sets Released for Planning
Invoke Plan Server to Retrieve Target Sets
Common Plan Representation
Target Alerter
Targets Written to Intermediate Tables in APS DB
DAOC Services
Replication of all operations to other instances of APS DB
Pull from Intermediate Tables into ATO-specific Tables
APS
APS DB
Intermediate Tables
ATO-specific Tables

Figure 3. Target Alerter Interfaces

The Mission Publication utility is the mechanism by which the missions planned using DAOC are created in the JFACC CPR. It can be run as an independent utility, but is more usually invoked automatically from the Air Battle Plan (ABP) Transfer tool, which transfers missions from the APS (plans) database to the FLEX (Operations (OPs)) database. Performing this transfer implies acceptance of the set of missions which have been planned, and so is a logical place to perform the additional step of publishing them to the CPR. The Orbix object references for targets and their objectives which were stored in the database by the Target Alerter are used by Mission Publication to link the new missions to the appropriate objectives in the CPR. In addition, the object references for the newly created mission objects are tracked in the FLEX database for future use by the Mission Publication or Update tools. If existing planned missions have been changed since the last time Publication was invoked, these changes are effected in the CPR. The Orbix object reference for each mission created or changed is placed in the Mission Event List with the event reason appropriately set. Other applications which are interested in these data can register interest in the Mission Event List and will be notified by a trigger when entries are made.

The Mission Update utility is used to publish mission status updates which have been entered through FLEX to the JFACC CPR. When a mission is updated, the Orbix object reference for it which has been stored in the FLEX database is used to retrieve the object from the JFACC CPR and modify it. The update is listed as an event in the Mission event list. Interfaces for the Mission utilities are illustrated in Figure 4 below.

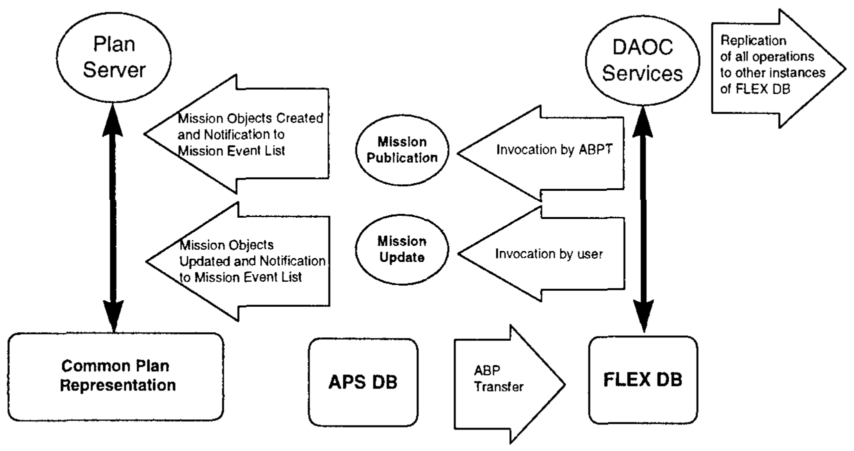
Plan Server
DAOC Services
Replication of all operations to other instances of FLEX DB
Mission Objects Created and Notification to Mission Event List
Mission Publication
Invocation by ABPT
Mission Objects Updated and Notification to Mission Event List
Mission Update
Invocation by user
Common Plan Representation
APS DB
ABP Transfer
FLEX DB

Figure 4. Mission Utility Interfaces

The overall design of the JJS DAOC planning system was driven by the capabilities of the JTF ATD services, particularly the Plan Server and the Data Server. The Plan Server has no query capability, making it impossible to port the applications from the Relational Database Management System (RDBMS) they currently use to the Plan Server. There is no coupling between the Data Server and the Plan Server, so mapping the CPR objects to the application databases was also not an option. The solution of bridging the gap with a series of data transfer utilities was a response to the shortcomings of the JTF ATD services and is not considered a complete solution to the problem of integrating legacy systems into the JFACC environment. A complete migration path for legacy applications into the JFACC environment requires expansion of the capabilities of the JTF ATD services, notably coupling of the Data Server with the Plan Server so that a legacy application could be integrated into the CPR without the need for recoding its database operations.

In addition to the planning capabilities of DAOC which were integrated into the JJS environment, a new intranet data mining application was developed. The DSS is an intranet application, accessible through any Java-enabled web browser (e.g., Netscape or Microsoft Internet Explorer) which is capable of performing dynamic queries on the application databases based on user input and of displaying the results of these queries in graphical format. The user is able to narrow the scope of the queries to the desired data through a filtering mechanism, and then is able to select from a list of pre-defined queries based on the functions requested. The data is presented in aggregated format, e.g., as a bar graph showing the status of all missions flying from a selected set of units (see Figure 5 below.) The purpose of presenting the data in this fashion is to allow the user to detect patterns which may be indicative of problems and which would otherwise take days of analysis to determine. The fact that it runs as a web-based application allows any user anywhere with a WAN connection to the DSS server to access data in the application databases directly.

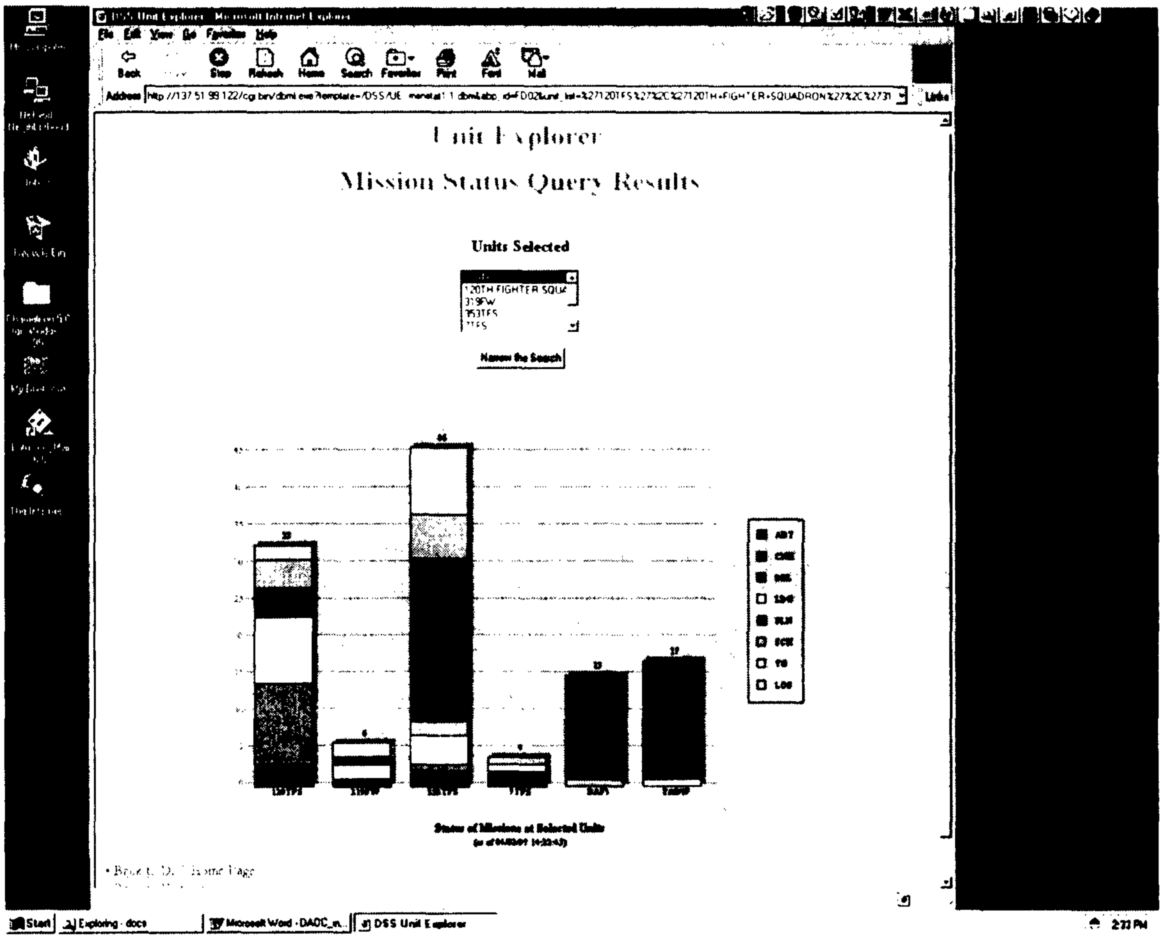

Figure 5. Results of a DSS query

The following figure details the filters and queries available for each of the explorers of the DSS. Each explorer provides a different focus for the data being examined, e.g., a mission status query made from the Unit Explorer would give the status of all missions flying from the selected units, a mission status query from the Objectives Explorer would give the status of all missions supporting the selected objectives, and a mission status query from the Mission Explorer would simply give the status of all the individually selected missions. Before presenting a selection list of objects to query on, the user has the opportunity to pare down the selection list by filtering on characteristics of the objects to be selected. For example, before selecting a set of units to query on in the Unit Explorer, the user may specify that only those units which have a particular type of aircraft, or those housed at certain bases. After a query has been run, the user may select a subset of the selected objects and re-run the query. This can be repeated an arbitrary number of times. Some queries also provide alternative ways to view the data and/or links to other explorers. An example of the latter: The user has selected a mission status query in the Unit Explorer. The user notices several canceled missions, and wishes to obtain additional information about those missions. The user can then link to the Mission Explorer with the same set of missions currently displayed in the Unit Explorer, select only the missions of interest, and then have access to all the queries of the Mission Explorer on that set of missions only.

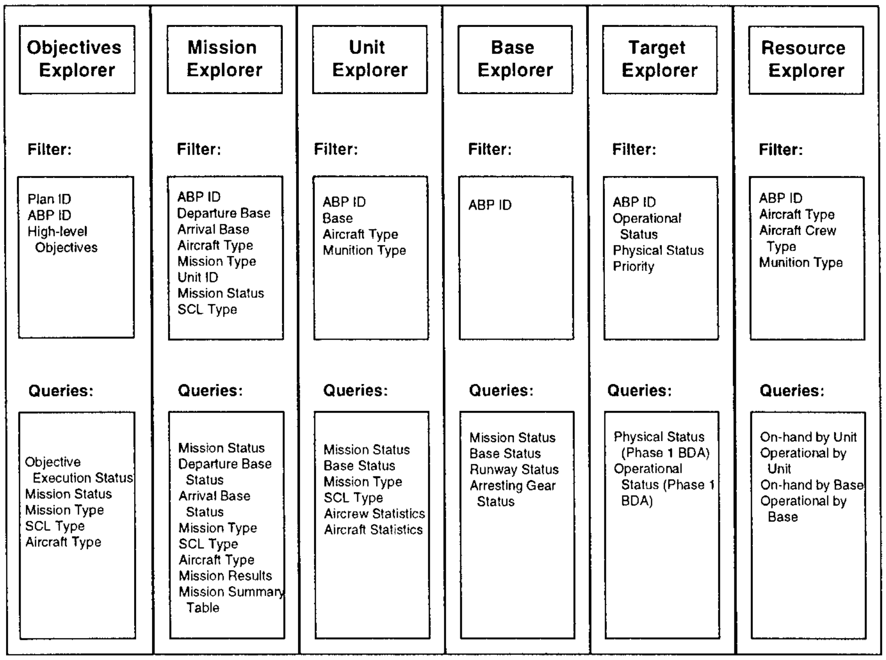

Figure 6. DSS Structure

The DSS server is PC-based and currently runs Windows '95 but can easily be upgraded to Windows NT. A collection of Commercial-Off-The-Shelf (COTS) products are used to implement the interface between the web browser and the application databases (which still reside on Suns.) SQL*Net is used to locate the application databases, and Open Database Connectivity (ODBC) is used to provide pass-through Sequential Query Language (SQL) capability. Allaire's Cold Fusion is used to provide the interface between HTML and SQL, so that queries can be made and their results manipulated within the format of a web page. In addition, the inclusion of the Objectives Explorer to support the JJS strategy-to-task planning concept required an interface with the JTF ATD Plan Server. Since the Plan Server itself has no query capability, a data transfer tool was provided to read the objectives from the CPR into the FLEX database. The links from objectives to missions and targets are already maintained in the application databases due to the planning activities, so adding the additional objective data gives full query capability on missions grouped by the objectives which they support. The graphical display of data through the DSS is accomplished by Java applets. Figure 7 below illustrates the DSS architecture used in the Jumpstart demo.

The DSS (without the Objectives Explorer, which is JJS-specific) will be incorporated into standard (that is, non-JJS) DAOC. Its interaction with the application databases allows it to be useful outside of a JFACC context. Its capabilities will be expanded for JWID '97, primarily to allow it to interface with the plan in progress, allowing the user a preview of the ATO before it is released. After JWID '97, the DSS can be enhanced to cache the data against which it queries on the client side, thus enabling queries to be run against this local set of data in the event of communications failure.

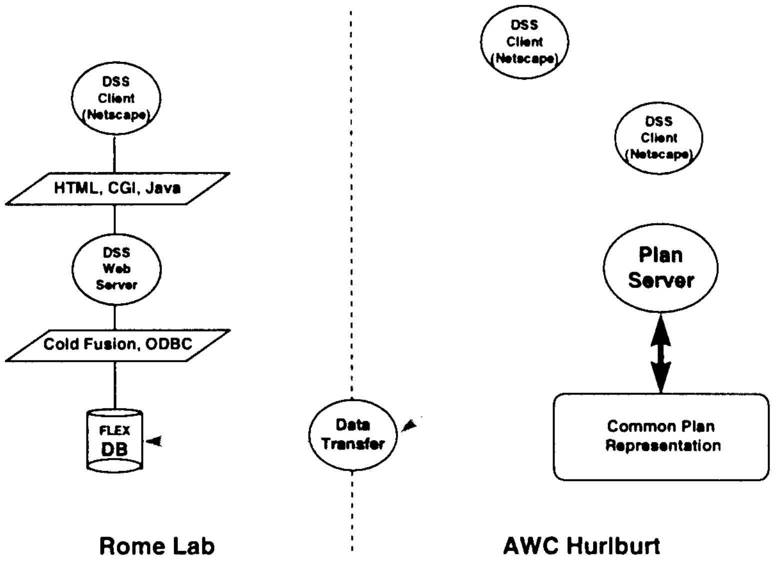

Figure 7. Architecture of DSS

3. Acronyms

ABP	Air Battle Plan
APS	Advanced Planning System
ATD	Advanced Technology Demonstration
ATO	Air Tasking Order
CDBAPI	Common Database Applications Program Interface
CORBA	Common Object Request Broker
COTS	Commercial-Off-The-Shelf
CPR	Common Plan Representation
CTAPS	Contingency Theater Automated Planning System
DAOC	Distributed Air Operations Center
DSS	Decision Support System
DUS	Distributed Update Server
FLEX	Force Level Execution System
GCCS LES	Global Command and Control System Leading Edge Services
JFACC	Joint Forces Air Component Command
JJS	JFACC Jumpstart
JOE OP	Joint Objectives Editor/Operations Planner
JTF	Joint Task Force
JWID	Joint Warrior Interoperability Demonstration
ODBC	Open Database Connectivity
OPs	Operations
ORB	Object Request Broker
OSA	Objectives/System Analysis
RAAP	Rapid Application of Air Power
RDB	Reusable Database
RDBMS	Relational Database Management System
SQL	Sequential Query Language
TBMCS	Theater Battle Management Core Systems
TNL	Target Nomination List